2018 年度 修士論文

グラフ書換え言語 LMNtal の索引付けと動的なパターンマッチングによる高速化

提出日： 2019 年 2 月 1 日
指導： 上田 和紀 教授

早稲田大学大学院 基幹理工学研究科
情報理工・情報通信専攻

学籍番号：5117F090-8
柳川 峻広

概要

柳川峻広

グラフ構造は，リストや木などのデータ構造，実社会における鉄道の路線や SNS のフォロー関係までさまざまな接続構造が表現できる．LMNtal とはそれらグラフ構造を直観的に扱って計算ができるグラフ書換え系である．LMNtal プログラムはグラフと書換えルールで構成され，書換えルールのパターンにマッチする部分グラフがあれば書換えが行われて処理が進んでいく．パターンにマッチする部分グラフを全体のグラフから高速に探索することは，実行性能を上げる上で重要である．

LMNtal にはハッシュテーブルのような Key Value Store の機能がない．既存の LMNtal の実行時処理系 SLIM は非連結な部分グラフを ID で紐づけるようなパターンマッチングの際でも，同種類のアトムのデータを 1 つ 1 つシーケンシャルアクセスして探索している．

また，SLIM は LMNtal コンパイラが出力する中間命令列を解釈実行する．そのため，書換えルール適用時の部分グラフを探索する手順は静的に決まる．しかし，静的にグラフパターンの探索の最適な手順を決定することは難しい．探索手順が悪いと探索途中で失敗してバックトラックする回数が増え，実行時間に大きく影響を与えることがある．

本論文ではこうしたパターンマッチングの効率を改善させるため，2 つの高速化手法を提案する．1 つ目は索引付けによる非連結なグラフ間のパターンマッチングの高速化手法，2 つ目は実行時のグラフの情報を元に動的にパターンマッチングの探索手順を決定することでバックトラックの回数を減らす手法である．

動的にパターンマッチが行えるインタプリタ Lmint を実装し，SLIM と性能比較をしたところ，本手法により時間計算量を減らし，バックトラックの回数も削減されることが確認できた．

Abstract

Takahiro Yanagawa

Graph structures can express various connection structures, such as lists and trees, railway routes in the real world and follow-up relationship of SNS. LMNtal is a graph rewriting system that allows us to manipulate these graph structures intuitively. An LMNtal program consists of graphs and rewrite rules, and if there is a subgraph that matches the pattern of a rewrite rule, rewriting is performed and computation proceeds. Searching a subgraph matching a pattern efficiently from the entire graph becomes important in improving execution performance.

LMNtal does not have a key value store function like a hash table. The existing runtime processing system SLIM of LMNtal searches for data of the same kind one by one by sequential access even in case of pattern matching that associates disjoint partial graphs with IDs.

In addition, SLIM interprets and executes the intermediate instruction sequence generated by the LMNtal compiler. Therefore, the procedure for searching a partial graph upon the application of a rewrite rule is statically determined. However, it is difficult to statically determine the optimal procedure for searching graph patterns. If the search procedure is bad, the number of backtracks that occurs during the search increases, which may greatly affect the execution time.

In this thesis, in order to improve the efficiency of such pattern matching, we propose two speeding up methods. The first method is to speed up pattern matching between disjoint graphs by indexing. The second is to reduce the number of backtracks by dynamically determining the procedure of searching for pattern matters based on the information of the graph at runtime.

By implementing the interpreter named Lmint which perform pattern matching dynamically and by comparing its performance with SLIM, we confirmed that our method reduces the time complexity and the number of backtracks.

目次

図目次

第 1 章

はじめに

1.1 研究の背景と目的

グラフ構造は，プログラミング言語における変数と値の関係，リストや木などのデータ構造，データベースにおけるリレーションから，実社会における鉄道の路線や SNS のフォロー関係までさまざまな接続構造が表現できる．LMNtal とはそれらグラフ構造を直観的に扱って計算ができるグラフ書換え系である．LMNtal プログラムはグラフと書換えルールで構成され，書換えルールのパターンにマッチする部分グラフがあれば書換えが行われて処理が進んでいく．パターンにマッチする部分グラフを全体のグラフから高速に探索することは，実行性能を上げる上で重要である．

LMNtal にはハッシュテーブルのような Key Value Store の機能がない．既存の LMNtal の実行時処理系 SLIM は非連結な部分グラフを ID で紐づけるようなパターンマッチングの際でも，同種類のアトムのデータを 1 つ 1 つシーケンシャルアクセスして探索している．

また，SLIM は LMNtal コンパイラが出力する中間命令列を解釈実行する．そのため，書換えルール適用時の部分グラフを探索する手順は静的に決まる．しかし，静的にグラフパターンの探索の最適な手順を決定することは難しい．探索手順が悪いと探索途中で失敗してバックトラックする回数が増え，実行時間に大きく影響を与えることがある．

本論文ではこうしたパターンマッチングの効率を改善させるため，2 つの高速化手法を提案する．1 つ目は索引付けによる非連結なグラフ間のパターンマッチングの高速化手法，2 つ目は実行時のグラフの情報を元に動的にパターンマッチングの探索手順を決定することでバックトラックの回数を減らす手法である．

本研究では，LMNtal プログラムを中間命令列へ静的にコンパイルする既存の方式では

なく，動的にパターンマッチングが行えるインタプリタ Lmint を設計し，実装を行った．SLIM と Lmint で性能比較をしたところ，本手法により時間計算量のオーダーを減らし，バックトラックの回数も削減されることが確認できた．

1.2　本論文の構成

本論文の構成は下記の通りである．第 2 章ではグラフ書換え言語 LMNtal について解説する．第 3 章では索引付けによる高速化手法について述べる．第 4 章では，動的なパターンマッチングによる高速化手法ついて述べる．第 5 章では例題を用いて第 3 と第 4 章で述べた高速化手法を実装した処理系と既存の処理系で性能比較を行う．第 6 章ではまとめと今後の課題に関して述べる．

第 2 章

LMNtal

本章ではグラフ書換え言語 LMNtal について紹介する．本論文では，LMNtal から膜を除いた Flat LMNtal を LMNtal として扱うことにする．また，型は unary と int のみを扱う．

解説するにあたり本章は文献 [1][3][4] をもとに再構成している．

2.1 LMNtal の概要

LMNtal はグラフの多重集合書換えに基づくプログラミング言語である．その名は Linked Multisets of Nodes transformation language に由来する．LMNtal は**アトム**，**リンク**，**ルール**の基本構成要素からなる．LMNtal ではグラフ理論 [6] におけるノードをアトム，エッジをリンクとしてグラフを表現する．ルールとは部分グラフに対する書換え規則である．LMNtal プログラムはグラフとルールで構成され，部分グラフにルールが適用されて書換えが起こり，処理が進んでいく．

2.2 LMNtal の構文

LMNtal の構文規則をまとめたものを図 2.1 に示す．X_i はリンク名, p はアトム名, P はプロセス, T はプロセステンプレートである．

LMNtal ではアトムとリンクでプロセスと呼ばれる無向多重グラフを表現する．プロセステンプレートはルール上で現れるグラフパターンを表す．アトムは LMNtal プログラム中では小文字から始まる文字列で表現され，リンクは大文字から始まる文字列で表現される．$p(X_1, \ldots, X_m)$ と書くと，p がアトム，$X_1, \ldots, X_m$ がそのアトムに接続してい

P	::=	$\mathbf{0}$		(空)
	|	$p(X_1, \ldots, X_m)$	$(m \geq 0)$	(アトム)
	|	P, P		(分子)
	|	T :- T		(ルール)
T	::=	$\mathbf{0}$		(空)
	|	$p(X_1, \ldots, X_m)$	$(m \geq 0)$	(アトム)
	|	T, T		(分子)

図 2.1　LMNtal の構文規則

るリンクを表す．また，省略構文として，$p(A_1, \ldots, A_m)$, $q(B_1, \ldots, B_i, \ldots, B_m)$ は A_m と B_i が同じリンクならば $q(B_1, \ldots, B_{i-1}, p(A_1, \ldots, A_{m-1}), B_{i+1}, \ldots, B_m)$ と略記して良い．

ルールは左辺のグラフパターンを右辺のグラフパターンに書き換える書換え規則である．拡張として *Guard* と呼ばれる制約条件をもったルール構文を次に示す．

$$Head \ \texttt{:-} \ Guard \ \texttt{|} \ Body \ .$$

この Guard には型制約や比較演算，四則演算を記述することができる．それらの一覧を図 2.2 に示す．unary 型は引数が 1 つのアトムのことであり，int 型は unary 型でかつ整数のアトムを指す．

プロセス P に 1 回だけ出現するリンクを P の自由リンクと呼び，P にちょうど 2 回出現するリンクを P の局所リンクと呼ぶ．また，プロセスには「同じリンク名が 2 回を超えて出現してはならない」というリンク条件がある．ただし，Guard に現れる型のあるリンクはこのリンク条件を無視してコピーや削除が行われて良い．

制約の種類	ガードに記述可能な制約	制約の意味	本成約が含意する型制約
型制約	`unary`(X)	X が unary 型である	
	`int`(X)	X が int 型である	`unary`(X)
比較制約	X_1 `==` X_2	$X_1 = X_2$	`unary`(X_1), `unary`(X_2)
	X_1 `\==` X_2	$X_1 \neq X_2$	
	X_1 `=:=` X_2	$X_1 = X_2$	`int`(X_1), `int`(X_2)
	X_1 `=\=` X_2	$X_1 \neq X_2$	
	X_1 `<` X_2	$X_1 < X_2$	
	X_1 `=<` X_2	$X_1 \leq X_2$	
	X_1 `>` X_2	$X_1 > X_2$	
	X_1 `>=` X_2	$X_1 \geq X_2$	
四則演算	X_1 `+` X_2	$X_1 + X_2$	`int`(X_1), `int`(X_2)
	X_1 `-` X_2	$X_1 - X_2$	
	X_1 `*` X_2	$X_1 * X_2$	
	X_1 `/` X_2	X_1/X_2	
	`+` X_2	$+X_2$	
	`-` X_2	$-X_2$	
	X_1 `mod` X_2	X_1 mod X_2	

図 2.2　LMNtal の Guard 制約

2.3　LMNtal 実行時処理系 SLIM

現在の実行時処理系である **SLIM** は C++ で実装されている．

LMNtal プログラムが実行される流れを図 2.3 に示す．最初に LMNtal プログラム (*.lmn) は **LMNtal コンパイラ**によってコンパイルされて中間命令列 (*.il) を生成する．そして**実行時処理系 SLIM** はこの中間命令列の解釈実行を行う．他にも LMNtal に関連した多数のアプリケーションが存在し，処理系を含めてこれらは GitHub にてソースコードが公開されている[*1]．

[*1] http://github.com/lmntal

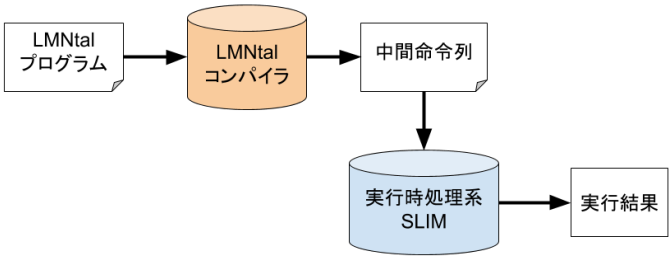

図 2.3　コンパイラと SLIM

2.4　SLIM におけるデータの管理

SLIM におけるグラフデータやルールの管理について各基本構成要素ごとに説明する．

2.4.1　アトムの管理

アトムの引数の個数を**価数** (arity) といい，価数が n のアトムを **n 価のアトム**という．また，アトム名と価数の組を**ファンクタ**と呼ぶ．アトム名 p，価数 n のファンクタを p/n と表す．SLIM はアトムのデータを同種類のファンクタごとに連結リストで管理しており，それぞれのアトムはリンク先のアトムをポインタでもつことでグラフを管理している．

2.5　中間命令

本節では，コンパイラが生成した中間命令列とそれによるルール実行について説明する．

2.5.1　ルール実行

あるルールが適用できるかどうか検査することを**パターンマッチング**という．ルールのグラフパターン内で局所リンクにより連結したアトムとリンクの集合を**連結パターン**と呼ぶ．SLIM は findatom 命令という中間命令によって，ルールの $Head$ の連結パターン内の始点となる 1 つのアトムからパターンの検査を行う．その対象となるアトムを**始点アトム**と呼ぶ．findatom 命令の引数に指定されたファンクタからハッシュ値計算によりアトムリストを参照し，その先頭のアトムから検査する．その後の中間命令で失敗した場合，1 つ前の findatom 命令までバックトラックし，アトムリスト内の次のアトムで再検

査する．アトムリスト中の末尾にあるアトムのパターンマッチングが失敗した場合，そのfindatom 命令はバックトラックを起こす．あるルールの中間命令の検査が全て成功し，commit 命令まで辿り着いたとき，そのルールのパターンマッチングは成功したことになり，ルールが適用される．逆に検査が全て失敗したとき，そのルールのパターンマッチングは失敗したことになる．

2.5.2　代表的な中間命令

パターンマッチングで使われる代表的な中間命令をいくつか説明する．各中間命令は，命令名と引数のリストからなる．引数として現れるものは，変数番号，変数番号のリスト，引数位置，ファンクタなどである．ファンクタはアトム名と価数を下線で繋いだ形，たとえば 3 価アトム `hoge` の場合は`'hoge'_3` という形で記述する．

- `findatom` [$dstatom$，$srcmem$，$funcref$]
  膜 $srcmem$ のファンクタ $funcref$ のアトムを 1 つ取得し，そのアトムへの参照を $dstatom$ に代入する．Flat LMNtal の場合，膜はなく $srcmem$ は常に 0 である．後続の命令列が失敗すると本命令までバックトラックが起き，別のアトムを取得して後続の命令列を再度実行する．
- `commit` [$rulename$，$lineno$]
  トレース用にルール名 $rulename$ を文字列で保持し，デバッガ用にソース上の行番号 $lineno$ を整数で保持する．この命令まで辿り着いたとき，ルール適用が成功したことになる．
- `derefatom` [$dstatom$，$srcatom$，$srcpos$]
  アトム $srcatom$ の第 $srcpos$ 引数のリンク先のアトムへの参照を $dstatom$ に代入する．
- `getfunc` [$func$，$atom$]
  アトム $atom$ のファンクタへの参照を $func$ に代入する．
- `func` [$srcatom$，$funcref$]
  アトム $srcatom$ がファンクタ $funcref$ を持つことを確認する．
- op [$dstintatom$，$intatom1$，$intatom2$]
  整数アトム $intatom1$ と $intatom2$ について，$intatom1$ op $intatom2$ の計算結果の整数アトムを生成し，$dstintatom$ に代入する．op には具体的に iadd（+），isub（−），imul（×），idiv（÷），imod（mod）がある．idiv および imod に限り 0 除

算があったとき失敗する．ineg の場合，*intatom*1 がなく，− *intatom*2 を計算結果として使う．

- *comp* [*intatom*1，*intatom*2]
  整数アトム *intatom*1 と *intatom*2 について，*intatom*1 *comp* *intatom*2 の関係が成り立つことを確認する．*comp* には具体的に ilt (<)，ile (≤)，igt (>)，ige (≥)，ieq（=)，ine（≠）がある．
- `isint` [*atom*]
  アトム *atom* が int 型であることを確認する．

第 3 章

索引付け

本章では 1 価のアトム (unary) の索引付けによる高速化手法について述べる．

3.1 既存の SLIM の問題点

LMNtal では，しばしばアトムに ID をつけるプログラムが書かれる．図 3.1 にその例を示す．このルールは order(ID, ProductID, CustomerID) に対して，ProductID と ID1 が一致する product(ID1, Name1, Location1)，および CustomerID と ID2 が一致する customer(ID2, Name2, Location2) にマッチする．

ファンクタ $functor$ のアトムリストを $L(functor)$ と表す．また，そのアトムリストの長さを $|L(functor)|$ と表す．既存の処理系の場合，このルールのマッチングは図 3.2 のような中間命令列に従う．まず，L(order/3) から 1 つアトムを取得して，その引数の型を検査する．次に L(product/3) から 1 つアトムを取得して，その引数の型を検査し，1 引数目が order(ID, ProductID, CustomerID) の 2 引数目と同じ値であるか確かめる．そうでなければバックトラックし，合っていれば次へ進む．最後に L(customer/3) から 1 つアトムを取得して，その引数の型を検査し，1 引数目が order(ID, ProductID, CustomerID) の 3 引数目と同じ値であるか確かめる．ここまでの検査に成功した 3 組のアトム order(ID, ProductID, CustomerID), product(ID1, Name1, Location1), customer(ID2, Name2, Location2) でパターンマッチング成功となる．以上の操作に従うと，このルール 1 回のパターンマッチングにかかる最悪時間計算量は $O(|L(\text{order/3})| * |L(\text{product/3})| * |L(\text{customer/3})|)$ である．

一般に，このような ID を使ったデータの管理にはハッシュテーブルやツリーなどの Key Value Store のコレクションが使われる．しかし，LMNtal にはそのようなコレク

```
order(ID, ProductID, CustomerID),
product(ID1, Name1, Location1), customer(ID2, Name2, Location2)
:-
ProductID =:= ID1, CustomerID =:= ID2,
Location1 >= 0, Location2 >= 0, ID >= 0
|
order_location(Location1, Location2),
product(ID1, Name1, Location1), customer(ID2, Name2, Location2).
```

図 3.1　ID を使うルール

ションやランダムアクセスが可能な配列などの機能はない．そのため ID を扱うようなプログラムでも，パターンマッチングの際にアトムリストをシーケンシャルに走査することになり，パターンの探索がボトルネックとなる．

3.2　1 価のアトムの索引付け

前節で述べた問題点の改善のため，1 価のアトム (unary) の索引付けによって Guard の等号制約を用いたパターンマッチングを高速化する手法を提案する．前節の例では，`order(ID, ProductID, CustomerID)` の `ProductID` と 1 引数目が一致する `product(ID1, Name1, Location1)` を探すのに L(`product/3`) を線形に走査していた．この `product/3` のアトムのデータが 1 引数目を key にした索引付けがされていれば，`ProductID =:= ID1` にを満たす `product(ID1, Name1, Location1)` をそのアトムがある限り失敗せずに取得できる．`customer(ID2, Name2, Location2)` についても同様である．

具体的な管理方法について説明する．unary を引数に持つアトムは，(ファンクタ, 引数の位置, unary の値) の組ごとに索引付けされたアトムリストで管理する．ファンクタと引数の位置を平衡二分木で索引付けし，その下の階層で unary の値をハッシュテーブルによって索引付けする．これはファンクタと引数の位置の種類は少なく，unary の値の種類は多いことから，平衡二分木とハッシュのオーバーヘッドの性質を考慮したためである．一回のハッシュ計算は重いが $O(1)$ であるため，サイズ N が大きくなるにつれて平衡二分木の対数時間 $O(log(N))$ に勝る．(ファンクタ, 引数の位置, unary の値) で索引付けさ

```
...
findatom     [1, 0, 'order'_3]
derefatom    [10, 1, 0]
isint          [10]
derefatom    [6, 1, 2]
isint          [6]
derefatom    [4, 1, 1]
isint          [4]
findatom     [2, 0, 'product'_3]
derefatom    [8, 2, 2]
isint          [8]
derefatom    [5, 2, 0]
isint          [5]
ieq            [4, 5]
findatom     [3, 0, 'customer'_3]
derefatom    [9, 3, 2]
isint          [9]
derefatom    [7, 3, 0]
isint          [7]
ieq            [6, 7]
commit         ["_orde", 0]
...
```

図 3.2　ID を使うルールの中間命令列

れたアトムリストを L(ファンクタ,引数の位置,$unary$ の値) で表す.

ただし，探索の始点に選ぶときを考慮して，通常のアトムリストでも管理する．上記の例の場合，ProductID が決まって product(ID1, Name1, Location1) を探索するときに通常の product/3 のアトムリストからではなく，(product/3, 1, $ProductID$) で索引付けされたアトムリストからアトムを取得することで，失敗することなくアトムを取得することができる．これにより，元のアトムリストのサイズに対して索引付けされたアトムリストのサイズが小さければ小さいだけ効率が上がる.

第 4 章

動的なパターンマッチング

4.1 静的なパターンマッチング手順決定の問題点

4.1.1 連結パターン内の始点

パターンマッチングにおいて，始点アトムをどれにするかによって探索の効率が大きく変わることがある．例えば，次のような append のプログラムを考える．

```
ans = append(L1,L2).
L1 = c(x1,c(x2,c(x3,nil))).
L2 = c(x4,c(x5,nil)).

c(V,Next,X), append(X,List,Pre) :- c(V,X,Pre), append(Next,List,X).
nil(X), append(X,List,Pre) :- Pre = List.
```

append/3 と c/3 を比べると c/3 は数が多いため，これを始点に選ぶとバックトラックの回数が増えてしまう．append/3 のような操作や関数を表すアトムを**関数的アトム**と呼び，c/3 のようなデータを表し，関数的なアトムと比べて個数が多くなるアトムを**データ的アトム**と呼ぶ．連結パターンのうち findatom で始点アトムとするアトムは関数的アトムを選ぶのが望ましい．しかし，LMNtal プログラムのルール記述から関数的アトムを静的に解析することは難しい．

4.1.2　複数の連結パターンの選択順序

ルールの連結パターンが複数あるとき，どの連結パターンを優先的に選ぶかによってパターンマッチングの効率が変わることがある．関数的アトムとデータ的アトムが非連結な場合，関数的アトムを優先的に選びたい．ある計算フェーズにおいてその関数的アトムがなければそのルールは即座に失敗できると，データ的アトムのアトムリストを走査しなくて済む．また，非連結な関数的アトムとデータ的アトムを ID による等号制約によってマッチさせるルールの場合，先に関数的アトムを選び，データ的アトムを 3 章の手法による索引付けされたアトムリストから取得することによって走査するアトムリストの長さを縮めることができる．

4.1.3　複数の索引対象の優先度

3 章の索引付けの対象となる箇所が複数現れる場合，索引付けされたアトムリストが小さい方を優先すべきである．次のルールを考える．

```
a(Xa,Ya), b(Xb,Yb) :- Xa=:=Xb, Ya=:=Yb| ...
```

いま a(Xa,Ya) がすでにマッチしているとする．このとき，通常のアトムリスト L(b/2) と索引付けされた L(b/2,1,Xa) と L(b/2,2,Ya) のうちどれを検索対象にするかを考える．明らかに通常のアトムリストよりも索引付けされたアトムリストの方がサイズは小さくなる．また，索引付けされたアトムリストの unary の値の重複度に差があれば，重複が少ない方を選ぶ方が良い．

4.2　小さいアトムリストを優先する動的な探索

LMNtal コンパイラが中間命令を静的に出力するときに，関数的アトムをルールから検出することは難しく，グラフの状態を利用した効率化はできない．これを解決する方法として，グラフの全体の状態に応じてルールのパターンマッチングの探索順序を動的に決定する手法を提案する．LMNtal のパターンマッチングの高速化に関する先行研究 [5] ではユーザーが処理系の内部事情を知らなくても実行性能を悪化させないよう，LMNtal グラフの書換えに応じて最適な中間命令列を動的に作成・適用させることを課題としている．

始点アトムの候補が複数ある，すなわちアトムを取り出すアトムリストの候補が複数あ

るとき，アトムリストのサイズが最小のアトムを貪欲に始点アトムとして選択すれば，検査の回数は最大でもそのアトムリストのサイズに抑えられて，バックトラックの回数を減らすことができる．また，関数的アトムを機械的に優先して選ぶことができる．

しかし，SLIM は基本的に中間命令列に従ってパターンマッチングを行うため，探索順序を動的に決定することは困難である．静的にあり得る探索順序を全列挙することも考えられるが，findatom 命令の個数の階乗に比例した中間命令列が必要になり，現実的ではない．

そこで，動的にパターンマッチングを行う実行時処理系を新たに設計する．SLIM の findatom に対応する動的なパターンマッチングのための findatom のアルゴリズムの擬似コードを図 4.1 に示す．

引数の rule はルールそのものを表し，registers はマッチング中のアトムのデータを格納するレジスタである．head_atom_num(rule) は rule の Head のアトムの数を表す．matched_atom(registers, i) は registers の i 番目のアトムのデータを指し，未決定であれば NULL が返る．get_atomlist(rule, i) は rule の i 番目のアトムのアトムリストが返る．もし等号制約により引数が決まっていて，索引付けされたアトムリストが使えるのであれば，その中で最小のアトムリストが返る．set_atom_to_reg(rule, registers, atom, head_id) は，registers の head_id 番目に始点アトムとして atom を格納したときにルール通りの連結パターンになっているか，ガード条件が満たされるかをチェックする bool 型の関数である．もし失敗したときはそこで格納した atom は取り除かれる．splice(StartList, iterator) はアトムリストである StartList の先頭から iterator の 1 つ前までをアトムリストの末尾に繋ぎ変える関数である．remove_atom_from_reg(rule, registers, atom, head_id) は atom が始点アトムとして成功しなかったときに呼ばれる関数で，registers の head_id 番目にある atom を取り除き，そこから繋がる連結パターンも registers から取り除く関数である．

あるルールを検査するとき，この findatom 関数を一度呼べばパターンマッチングに成功する場合は true が返り，レジスタにはマッチしたアトムや自由リンクが格納される．失敗したときは false が返り，そのルールが適用できないということになる．

```
function FIND_ATOM(rule, registers)
    StartList ← NULL
    head_id ← −1
    for each i ∈ {1, . . . , head_atom_num(rule)} do
        if matched_atom(registers, i) ≠ NULL then
            continue
            AtomList ← GET_ATOMLIST(rule, i)
            if StartList = NULL ∨ size(StartList) > size(AtomList) then
                StartList ← AtomList
                head_id ← i
            end if
        end if
    end for
    if StartList = NULL then
        return true
    else
        for each iterator ∈ StartList do
            atom ← ∗iterator
            if SET_ATOM_TO_REG(rule, registers, atom, head_id) then
                if FIND_ATOM(rule, registers) then
                    SPLICE(StartList, iterator)
                    return true
                else
                    REMOVE_ATOM_FROM_REG(rule, registers, atom, head_id)
                end if
            end if
        end for
        return false
    end if
end function
```

図 4.1　findatom のアルゴリズム

4.2.1　関連研究

関連研究として，LMNtal に近い Constraint Handling Rules(CHR)[7] という多重集合書換えに基づく制約プログラミング言語がある．LMNtal のパターンマッチングの探索の順序に対応するのが Join Ordering[8] である．CHR の制約ストアの統計情報からコストが小さい join order を求めるヒューリスティックが提案されていて，小さいアトムリストから選ぶ手法と類似している．本手法と同じ発想で，索引付けが利用できる場合のコストは小さく見積もる．しかし，本手法ではパターンマッチングの途中の結果によって探索順序が動的に変わるという点で異なる．

第 5 章

実験

この章では 3 章 4 章で挙げた提案手法を実現するために LMNtal インタプリタ **Lmint**(LMNtal minimum interpreter) を設計した．このソースコードは GitHub で公開されている．[*1]． 実装量は約 2000LOC 程度で，既存の処理系 (LMNtal コンパイラ,SLIM) が約 90000LOC と比較すると大幅に軽量化されている．実行時処理系 SLIM と Lmint で，いくつかの例題について実行効率を評価する．実行環境を表 5.1 に示す．

表 5.1　実験環境

CPU	Intel Core i3-6300T CPU @ 3.30GHz × 4
MEM	15.6GiB
LMNtal コンパイラ	version 1.44
LMNtal オプション	`--slimcode -O3`
SLIM	version 2.3.1
SLIM オプション	`-p2 --use-builtin-rule --no-dump`

5.1　共通フォロー

ソーシャルネットワーキングサービス (SNS) 上のフォロー関係について，次のような問題設定を考える．A さんが B さんをフォローしていることを `follow(`A`,`B`)` と表す．大規模なアカウントが 10 人いて，彼らは約 N 人をフォローし，またフォローされてい

[*1] http://github.com/lmntal/lmint

る．一方，小規模なアカウントは約 N 人いて，約 10 人をフォローしている．このとき，A さんと B さんどちらも共通でフォローしている人を列挙するクエリが 2 つ与えられる．1 つは A さんが大規模アカウントで B さんが小規模アカウントのとき，もう 1 つは A さんが小規模アカウントで B さんが大規模アカウントのときである．図 5.1 にそのルールを示す．付録 A.1 に示す．

```
common(A,B,L), follow(A1,C1), follow(B2,C2) :-
    A =:= A1, B =:= B2, C1 =:= C2 |
    common(A,B,X), c(C1,L,X).
```

図 5.1　共通フォローのルール

このプラグラムについて，N をスケールさせて SLIM と Lmint(探索順序固定, 動的) で実行時間およびバックトラックの回数を測定した．このプログラムにおいて探索順序固定の時の findatom の順番は `common(A,B,L)` → `follow(A1,C1)` → `follow(B2,C2)` である．実行時間の測定結果を図 5.2 に，バックトラックの回数の測定結果を図 5.3 に示す．lmint(static) は探索順序固定を表し，lmint は探索順序動的を表す．この結果から，実行時間のオーダーが $O(N^2)$ から $O(N)$ になっていることが分かる．索引付けの無い SLIM は 2 つの `follow/2` の組み合わせを探索するので $O(|L(\texttt{follow/2})|^2)$ すなわち $O(N^2)$ となる．索引付けがある Lmint は，どちらの common クエリに対しても 2 つの `follow/2` のうち 1 つは小規模アカウントからのフォローを表すので，アトムリストの線形走査が $O(N)$ から $O(1)$ となり，オーダーが良くなっている．また，SLIM と比べて Lmint のバックトラックの回数も大幅に減少している．Lmint の探索順序が固定のものと動的のものを比較すると，動的の方がバックトラックの回数が少ないことが確認された．

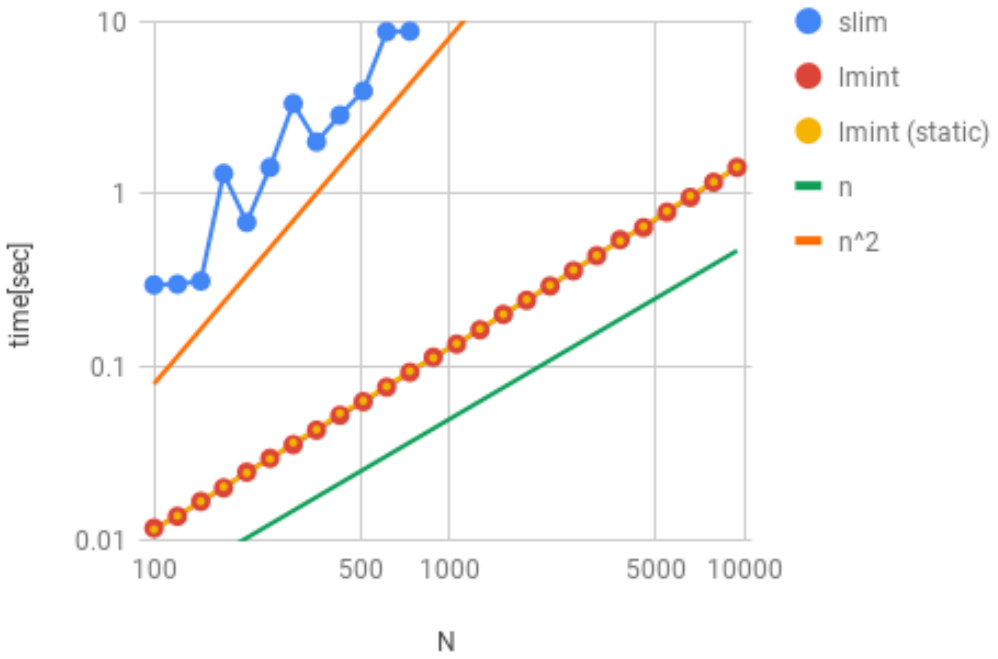

図 5.2　共通フォローの実行時間

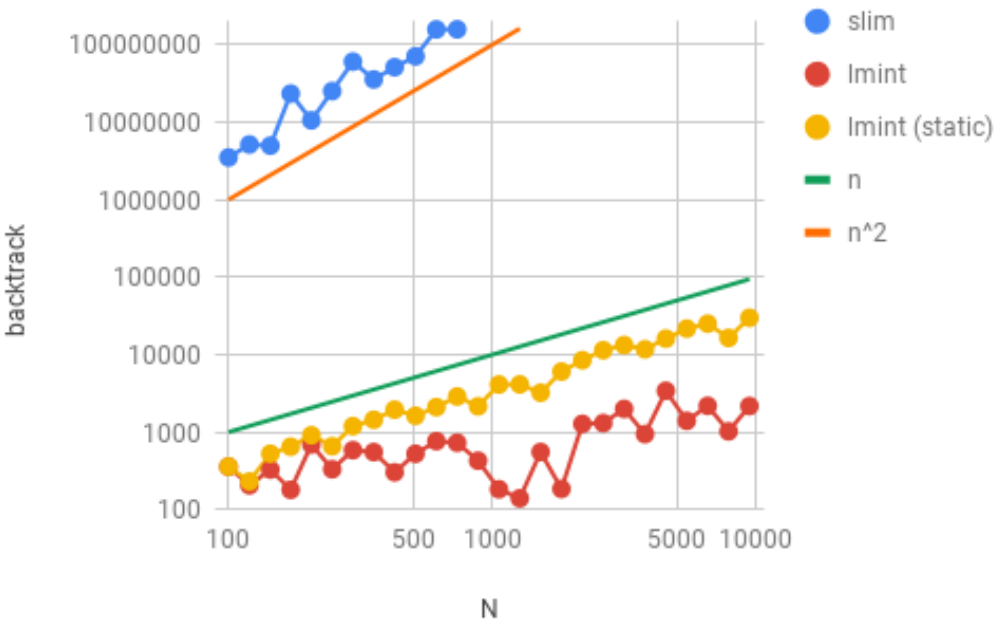

図 5.3　共通フォローのバックトラックの回数

5.2　RDB の join 演算

Relational Data Base (RDB) [9] において join とは異なるテーブルを結合する演算である．例を図 5.4 に示す．

図 5.4　RDB の join の例

join 演算である order のクエリを LMNtal のルールで表現したものを図 5.5 に示す．M 列のテーブル T の各行のデータを，ファンクタ T/M のアトムとその引数で表す．`order(ID, ProductID, CustomerID)` は join のクエリを表し，$ProductID$ と $CustomerID$ に一致する `product/3` と `customer/3` をそれぞれ探索する．

実験のため，次のような問題設定を考える．初期グラフとして $|L(\texttt{order/3})| = N$, $|L(\texttt{product/3})| = N$, $|L(\texttt{customer/3})| = 100$ のサイズでグラフが与えられる．全ての `order/3` のアトムに対して書換えが成功し，`order_location/3` が生成される．その初

期グラフを生成するプログラムを付録 A.2 に示す.

```
order(ID, ProductID, CustomerID),
product(ID1, Name1, Location1), customer(ID2, Name2, Location2)
:-
ProductID =:= ID1, CustomerID =:= ID2,
Location1 >= 0, Location2 >= 0, ID >= 0
|
order_location(ID, Location1, Location2),
product(ID1, Name1, Location1), customer(ID2, Name2, Location2).
```

図 5.5　RDB の join のルール

このプラグラムについて，N をスケールさせて SLIM と Lmint(探索順序固定, 動的) で実行時間およびバックトラックの回数を測定した．このプログラムにおいて探索順序固定の時の findatom の順番は `order(ID, ProductID, CustomerID)` → `product(ID1, Name1, Location1)` → `customer(ID2, Name2, Location2)` である．実行時間の測定結果を図 5.6 に，バックトラックの回数の測定結果を図 5.7 に示す．SLIM は $N = 3000$ 付近でセグメンテーションフォルトを起こした．実行時間のオーダーはどれも $O(N)$ になっていることが分かる．Lmint の探索順序固定のバックトラックの回数は全て 0 であったので，図のグラフには描かれていない．SLIM はバックトラックの回数が $O(N^2)$ であるのに対して Lmint はどちらも定数回で大幅に減少している．Lmint の探索順序が固定のものと動的のものを比較すると，探索順序固定は 0 回なのに対して，動的は約 100 回のバックトラックを起こしていた．探索順序が固定は `order/3` から最初に findatom するため必ずパターンマッチングに成功する．一方，動的は実行の終盤に差し掛かるまで `customer/3` を findatom してその 1 引数目の $ID2$ と 3 引数目が一致する `order(ID, ProductID, CustomerID)` がなくなるまで書換えが行われ，なくなるときにバックトラックして次の `customer/3` のアトムを取得する．およそ L(`customer/3`) を一巡するために約 100 回のバックトラックを起こしていると考えられる．このように問題設定によってはサイズの小さいアトムリストを貪欲に選ぶほうがバックトラックを多く起こしてしまうこともある．

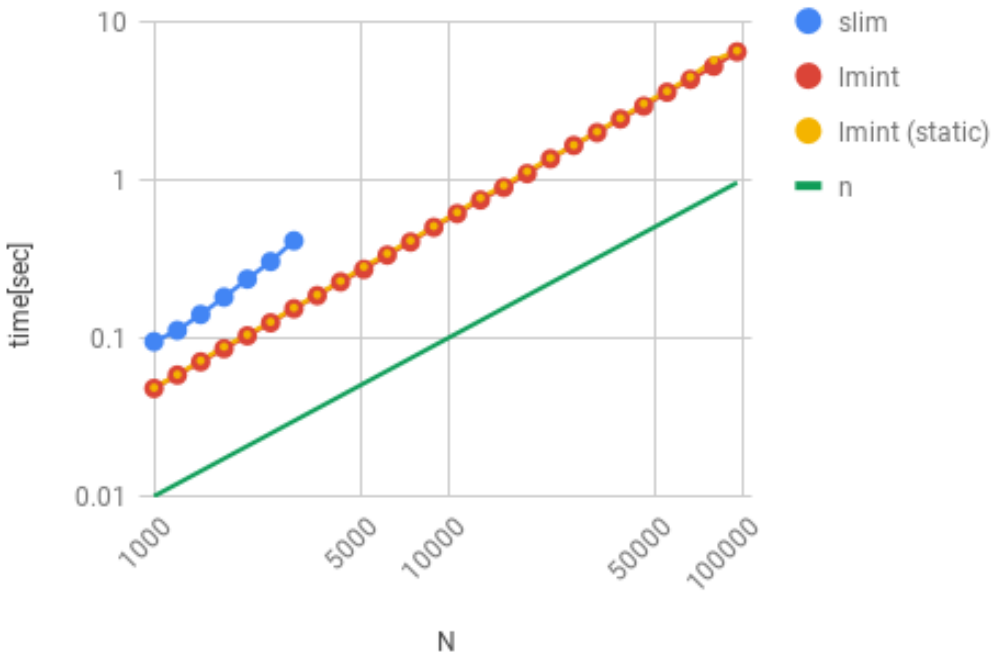

図 5.6　RDB の join の実行時間

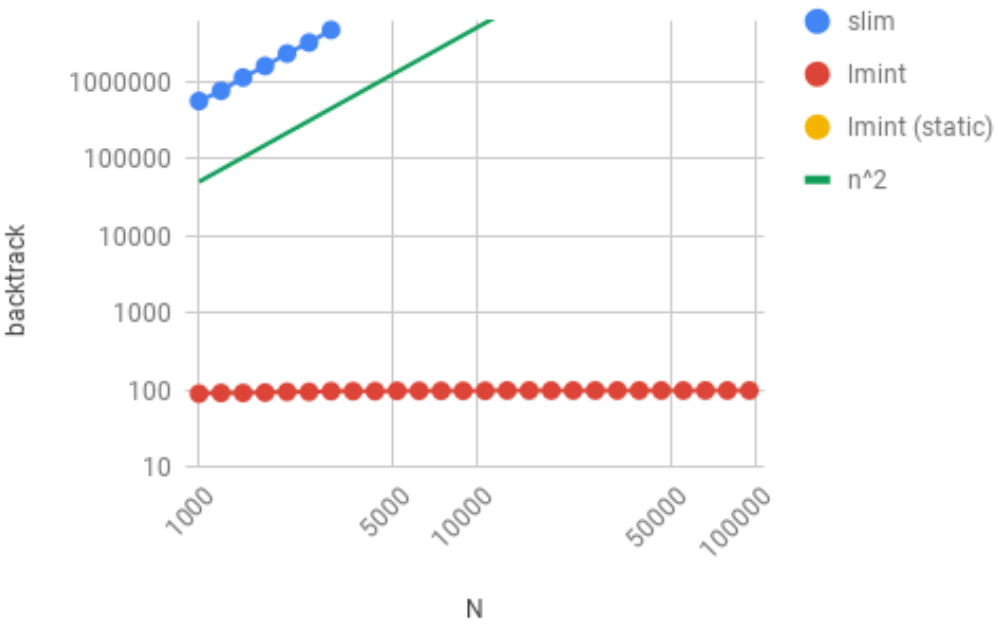

図 5.7　RDB の join のバックトラックの回数

第 6 章

まとめと今後の課題

6.1 まとめ

本研究では LMNtal のパターンマッチングの高速化手法について 2 つ提案した．1 つ目は 1 価のアトムを索引づけしてアトムリストを細分化することにより等号制約があるときに対象とするアトムリストの対象を狭める手法で，2 つ目はアトムリストのサイズが小さいアトムから貪欲に選ぶような動的なパターンマッチングを行う手法であった．これらを実装したインタプリタ Lmint と既存の実行時処理系 SLIM で性能評価を行なった．その結果，時間計算量のオーダーを下げ，バックトラックの回数も減らすことに成功した．しかし，問題設定によってはアトムリストのサイズが小さいアトムから貪欲に選ぶことにより，バックトラックの回数を増やしてしまう例もあった．

また，既存の LMNtal コンパイラおよび SLIM と比べて，インタプリタは処理系としての実装量が大幅に削減された．LMNtal コンパイラと SLIM は合わせて約 90000LOC あるのに対して，Lmint は Flat LMNtal の通常実行に絞ってはいるがわずか約 2000LOC である．

6.2 今後の課題

今後の課題としては，SLIM にある機能として非決定実行を実装が挙げられる．非決定実行は書換え可能なアトムの組み合わせについて全て検査する必要があり，走査するアトムリストの長さをなるべく縮める本研究による効率化が期待できる．また，Lmint の実装量が軽量であることを生かし，新たな機能を追加することも挙げられる．

謝辞

本研究を進めるにあたり，ご指導を賜わった上田和紀教授に深く感謝致します．そして，困っている時に研究の相談に乗っていただいた恒川先輩，分からないことがあったら何から何まで教えてくれてデバッグも手伝ってくれた冨岡くん，上手くいかないときに励まし合った増田健太くん，精神が不安定になったときに落ち着かせてくれた増田翔仁くん，例題探しのヒントをくれた齊藤くん，みなさま皆様本当にありがとうございました．最後に，入学から現在に至るまで，学生生活支えてくださった家族に感謝の意を表します．

2019 年 1 月　柳川 峻広

参考文献

[1] 上田和紀, 加藤紀夫: 言語モデル LMNtal , コンピュータ ソフトウェア Vol.21, No.2, pp.126-142, 2004.

[2] 乾敦行, 工藤晋太郎, 原耕司, 水野謙, 加藤紀夫, 上田和紀: 階層グラフ書換えモデルに基づく統合プログラミング言語 LMNtal , コンピュータソフトウェア, Vol.25, No.1, pp.124-150, 2008.

[3] 村山敬, 工藤晋太郎, 櫻井健, 水野謙, 加藤紀夫, 上田和紀: 階層グラフ書換え言語 LMNtal の処理系, コンピュータソフトウェア, Vol.25, No.2, pp.47-77, 2008.

[4] 石川力, 堀泰祐, 村山敬, 岡部亮, 上田和紀: 軽量な LMNtal 実行時処理系 SLIM の設計と実装, 情報処理学会第 70 回全国大会, 2008.

[5] 青山龍一: グラフ書換え系 LMNtal の実行時処理系 SLIM におけるグラフパターンマッチングの高速化, 修士論文, 早稲田大学大学院 基幹理工学研究科 情報理工学専攻, 2015.

[6] 茨木俊秀, 永持仁, 石井利昌: グラフ理論—連結構造とその応用—, 朝倉書店, 2010.

[7] Frühwirth, Thom: ”Theory and practice of constraint handling rules.” The Journal of Logic Programming, Vol.37, No.1-3, pp.95-138, 1998.

[8] De Koninck, Leslie, and Jon Sneyers: ”Join ordering for constraint handling rules.” Proceedings of the Fourth Workshop on Constraint Handling Rules. U. Porto, 2007.

[9] Mishra, Priti, and Margaret H. Eich: ”Join processing in relational databases.” ACM Computing Surveys (CSUR), Vol.24, No.1, pp.63-113, 1992.

外部発表

[1] 柳川峻広，上田和紀： グラフ書換え言語 LMNtal の実行時処理系 SLIM における制約付き部分グラフ探索の高速化，第 19 回プログラミングおよびプログラミング言語ワークショップ (PPL 2017)，ポスター発表，2017.

[2] 柳川峻広，上田和紀： グラフ書換え系言語 LMNtal におけるパターンマッチングの実行時最適化，情報処理学会第 80 回全国大会，2018.

Appendix: 初期グラフを生成するプログラム

第 5 章で例題の初期グラフを生成するプログラムを示す.

ソースコード A.1　共通フォローの初期グラフを生成するプログラム (C++)

```
#include <bits/stdc++.h>
using namespace std;

int main(void){
    int n;
    cin >> n;
    vector<set<int>> g(n);
    std::random_device seed_gen;
    std::mt19937 mt(seed_gen());
    std::uniform_int_distribution<> rand_int(0, n-1);

    // 大規模
    for (int i = 0; i < 10; i++) {
        for(int j = 0; j < n/10; j++) {
            int r = rand_int(mt);
            g[i].insert(r);
            g[r].insert(i);
        }
    }
    // 小規模
    for (int i = 10; i < n; i++) {
        for (int j = 0; j < 10; j++) {
            int r = rand_int(mt);
            g[i].insert(r);
        }
    }

    printf("common(3,13,nil). common(17,7,nil).\n");
    int cnt = 0;
    for (int i = 0; i < n; i++) {
        for (int to : g[i]) {
            printf("follow(%d,%d).\n", i, to);
        }
    }
    return 0;
}
```

ソースコード A.2　RDB の join の初期グラフを生成するプログラム (C++)

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(void){
    std::random_device seed_gen;
    std::mt19937 mt(seed_gen());
    int sizeO, sizeP, sizeC;
    int n;
    cin >> n;
    sizeO = n;
    sizeP = n;
    sizeC = 100;

    // Order
    for (int i = 0; i < sizeO; i++) {
        std::uniform_int_distribution<> rand_int(0, sizeP-1);
        int p = rand_int(mt);
        rand_int = std::uniform_int_distribution<>(0, sizeC-1);
        int c = rand_int(mt);
        printf("order(%d,%d,%d).\n", i, p, c);
    }

    std::uniform_int_distribution<> rand_int(0, 100000);
    // Product
    for (int i = 0; i < sizeP; i++) {
        printf("product(%d,%d,%d).\n", i, rand_int(mt), rand_int(mt));
    }
    // Customer
    for (int i = 0; i < sizeC; i++) {
        printf("customer(%d,%d,%d).\n", i, rand_int(mt), rand_int(mt));
    }

    return 0;
}
```

Appendix: Lmint のソースコード

第 3 章および第 4 章で提案した手法を実装した LMNtal のインタプリタ Lmint のソースコードである．GitHub リポジトリ (https://github.com/lmntal/lmint) にも公開している．

ソースコード A.3　parser.hpp

```
1  #ifndef PARSER_HPP
2  #define PARSER_HPP
3
4  #include <iostream>
5  using std::cin;
6  using std::cout;
7  using std::cerr;
8  using std::endl;
9  using std::ostream;
10 using std::getline;
11 #include <vector>
12 using std::vector;
13 #include <string>
14 using std::string;
15 using std::to_string;
16 #include <map>
17 using std::map;
18 using std::pair;
19 #include <algorithm>
20 using std::max;
21 using std::min;
22 #include <cassert>
23
24 #include "rule.hpp"
25
26 // EBNF of LMNtal parser
27 // Program := {(Rule | Graph) '.'}
28 // Rule := TopSet ':-' TopSet
29 // Graph := TopSet
30 // TopSet := (Nest | Nest '=' Nest) {',' (Nest | Nest '=' Nest)}
31 // Nest := Link | Atom | Atom '(' ')' | Atom '(' Nest {',' Nest} ')'
32 // Atom := [a-z]{[a-zA-Z_0-9]}
33 // Link := [A-Z_]{[a-zA-Z_0-9]}
34
35 class Parser;
36 class TopSet;
37 class Result_of_nest;
38 class PrsGuard;
39 class PrsRule;
40
41
```

```
42 class TopSet{
43   public:
44     vector<string> atom_names;
45     vector<vector<string>> atoms_args;
46     vector<pair<string,string>> connects;
47     map<string,string> dst;
48     map<string,int> link_count;
49
50     TopSet();
51     ~TopSet();
52
53     int add_atom(string atom_name);
54     static string get_uniq_local_link_name();
55     void build_dst_map();
56     void build_link_count();
57     void validate_links();
58     void show();
59 };
60
61
62 class PrsGuard{
63   public:
64
65     class Compare{
66       public:
67         vector<string> left_exp;
68         string op;
69         vector<string> right_exp;
70
71         Compare();
72         ~Compare();
73     };
74
75     class TypeCheck{
76       public:
77         string type;
78         string link_name;
79
80         TypeCheck();
81         ~TypeCheck();
82     };
83
84     class Assign{
85       public:
86         string new_var;
87         vector<string> exp;
88
89         Assign();
90         ~Assign();
91     };
92
93     vector<TypeCheck> type_checks;
94     vector<Compare> compares;
95     vector<Assign> assigns;
96
97     PrsGuard();
98     ~PrsGuard();
99
100    void show();
101 };
102
103
104 class PrsRule{
105   public:
106     TopSet head, body;
```

```
107     PrsGuard guard;
108     map<string, int> var_id;
109
110     PrsRule();
111     ~PrsRule();
112     PrsRule(TopSet head_, TopSet body_);
113     PrsRule(TopSet head_, PrsGuard guard_, TopSet body_);
114
115     void build_var_id();
116     void validate_links();
117     void show();
118 };
119
120
121 class Parser{
122   public:
123     int y,x;
124     vector<string> raw_inputs;
125     TopSet graph;
126     vector<PrsRule> rules;
127
128     Parser();
129     ~Parser();
130
131     void read_istream();
132     void syntax_error(string message);
133     void read_ignore();
134     bool read_token(string token);
135     void read_sentences();
136     TopSet read_topset();
137     Result_of_nest read_nest(TopSet &topset);
138
139     bool read_factor(vector<string> &tokens);
140     bool read_exp(vector<string> &tokens);
141     bool read_type_check(PrsGuard::TypeCheck &type_check);
142     bool read_assign(PrsGuard::Assign &assign);
143     bool read_compare(PrsGuard::Compare &compare);
144     bool read_guard(PrsGuard &guard);
145
146     string read_number();
147     string read_name();
148
149     void parse();
150     void show();
151 };
152
153
154 class Result_of_nest{
155   public:
156     bool is_link;
157     int atom_id;
158     string link_name;
159
160     Result_of_nest();
161     ~Result_of_nest();
162     Result_of_nest(string link_name_);
163     Result_of_nest(int atom_id_);
164 };
165
166
167 bool is_link_initial(char c);
168 bool is_atom_initial(char c);
169
170 // vm
171 void set_graph(TopSet &graph);
```

```
172 void set_rule(PrsRule &parser_rule);
173 void load(Parser &parser);
174
175 #endif
```

ソースコード A.4　parser.cpp

```
1  #include "parser.hpp"
2
3  TopSet::TopSet() {}
4  TopSet::~TopSet() {}
5
6  int TopSet::add_atom(string atom_name) {
7      atom_names.push_back(atom_name);
8      int atom_id = atoms_args.size();
9      atoms_args.resize(atom_id + 1);
10     return atom_id;
11 }
12
13 string TopSet::get_uniq_local_link_name() {
14     static int uniq_local_link_num = 0;
15     return "#" + to_string(uniq_local_link_num++);
16 }
17
18 void TopSet::show() {
19     for (int i = 0; i < (int)atom_names.size(); i++) {
20         cout << atom_names[i] << "(";
21         int arity = (int)atoms_args[i].size();
22         for (int j = 0; j < arity; j++) {
23             cout << atoms_args[i][j]
24                  << (j + 1 == arity ? "" : ", ");
25         }
26         cout << ")" << endl;
27     }
28     for (auto &p : connects) {
29         cout << p.first << " = " << p.second << endl;
30     }
31 }
32
33
34 PrsGuard::Compare::Compare() {}
35 PrsGuard::Compare::~Compare() {}
36
37 PrsGuard::TypeCheck::TypeCheck() {}
38 PrsGuard::TypeCheck::~TypeCheck() {}
39
40 PrsGuard::Assign::Assign() {}
41 PrsGuard::Assign::~Assign() {}
42
43 PrsGuard::PrsGuard() {}
44 PrsGuard::~PrsGuard() {}
45
46
47 void PrsGuard::show() {
48     for (TypeCheck &type_check : type_checks) {
49         cout << type_check.type << ": " << type_check.link_name << endl;
50     }
51
52     for (Compare &compare : compares) {
53         for (string &token : compare.left_exp) {
54             cout << token << " ";
55         }
56         cout << compare.op << " ";
```

```
57          for (string &token : compare.right_exp) {
58              cout << token << " ";
59          }
60          cout << endl;
61      }
62
63      for (Assign &assign : assigns) {
64          cout << assign.new_var << " = ";
65          for (string &token : assign.exp) {
66              cout << token << " ";
67          }
68          cout << endl;
69      }
70  }
71
72
73  PrsRule::PrsRule() {}
74  PrsRule::~PrsRule() {}
75
76  PrsRule::PrsRule(TopSet head_, TopSet body_): head(head_), body(body_) {}
77
78  PrsRule::PrsRule(TopSet head_, PrsGuard guard_, TopSet body_):
79      head(head_), body(body_), guard(guard_) {}
80
81  void PrsRule::show() {
82      printf(". [Head] .\n");
83      head.show();
84      printf(". [Guard] .\n");
85      guard.show();
86      printf(". [Body] .\n");
87      body.show();
88  }
89
90
91  Parser::Parser(): y(0), x(0) {}
92  Parser::~Parser() {}
93
94  void Parser::read_istream() {
95      string s;
96      while (getline(cin, s)) {
97          raw_inputs.push_back(s);
98      }
99  }
100
101 void Parser::show() {
102     printf("================ [Graph] ================\n");
103     graph.show();
104
105     printf("================ [Rule] =================\n");
106     int R = rules.size();
107     for (int i = 0; i < R; i++) {
108         printf("* ------- [Rule %d] -------- *\n", i);
109         rules[i].show();
110     }
111     printf("=========================================\n");
112 }
113
114
115 Result_of_nest::Result_of_nest() {}
116 Result_of_nest::~Result_of_nest() {}
117
118 Result_of_nest::Result_of_nest(string link_name_):
119     is_link(true), link_name(link_name_) {}
120
121 Result_of_nest::Result_of_nest(int atom_id_):
```

```
122     is_link(false), atom_id(atom_id_) {}
123
124
125 void Parser::syntax_error(string message) {
126     cerr << "Syntax Error: " << message
127          << " at line " << y+1 << " column " << x+1 << endl;
128     int l = max(x-30, 0);
129     cerr << raw_inputs[y].substr(l, 70) << endl;
130     cerr << string(x-l, ' ') << "^" << endl;
131     exit(1);
132 }
133
134
135 // https://stackoverflow.com/questions/6698039/nested-comments-in-c-c
136 void Parser::read_ignore() {
137     while (y < (int)raw_inputs.size()) {
138         if (x == (int)raw_inputs[y].size()) {
139             y++, x = 0;
140         } else if (raw_inputs[y][x] == ' ' || raw_inputs[y][x] == '\t') {
141             x++;
142         } else if (raw_inputs[y].substr(x, 2) == "//") {
143             y++, x = 0;
144         } else if (raw_inputs[y].substr(x, 2) == "/*") {
145             int begin_y = y, begin_x = x;
146             x += 2;
147             while (raw_inputs[y].substr(x, 2) != "*/") {
148                 if (x == (int)raw_inputs[y].size()) {
149                     y++, x = 0;
150                 } else {
151                     x++;
152                 }
153                 if (y == (int)raw_inputs.size()) {
154                     y = begin_y, x = begin_x;
155                     syntax_error("unterminated comment");
156                 }
157             }
158             x += 2;
159         } else {
160             return;
161         }
162     }
163 }
164
165
166 bool Parser::read_token(string token) {
167     int length = token.size();
168     if (raw_inputs[y].substr(x, length) == token) {
169         x += length;
170         read_ignore();
171         return true;
172     } else {
173         return false;
174     }
175 }
176
177
178 // Program := {(Rule | Graph) '.'}
179 // Rule := TopSet ':-' TopSet
180 // Graph := TopSet
181 void Parser::read_sentences() {
182     read_ignore();
183     while (y < (int)raw_inputs.size()) {
184         TopSet topset = read_topset();
185
186         // マクロにスコープはない？！ -> lambda 式
```

```
187         #define append(a, b) a.insert(a.end(), b.begin(), b.end())
188         // Graph
189         if (read_token(".")) {
190             append(graph.atom_names, topset.atom_names);
191             append(graph.atoms_args, topset.atoms_args);
192             append(graph.connects, topset.connects);
193         }
194         // Rule
195         else if (read_token(":-")) {
196             PrsGuard guard;
197             int preY = y, preX = x;
198             if (read_guard(guard)) {
199                 rules.push_back(PrsRule(topset, guard, read_topset()));
200             } else {
201                 y = preY, x = preX;
202                 rules.push_back(PrsRule(topset, read_topset()));
203             }
204             if (!read_token(".")) {
205                 syntax_error("Unexpected Token");
206             }
207         } else {
208             syntax_error("Unexpected Token");
209         }
210     }
211 }
212
213
214 /*
215     Link = atom, atom = Link ならatom側へ付け足し
216     Link = Link ならそのまま pair<string, string> と等価なもので登録
217     atom = atom なら局所リンクを発行し,それぞれ付け足し
218 */
219 // TopSet := (Nest | Nest '=' Nest) {',' (Nest | Nest '=' Nest)}
220 TopSet Parser::read_topset() {
221     TopSet topset;
222     do {
223         int tokenX = x, tokenY = y;
224         Result_of_nest result1 = read_nest(topset);
225         if (read_token("=")) {
226             Result_of_nest result2 = read_nest(topset);
227             if (result1.is_link) {
228                 // Link = Link
229                 if (result2.is_link) {
230                     topset.connects.push_back({result1.link_name, result2.link_name});
231                 }
232                 // Link = Atom(...)
233                 else {
234                     int atom_id = result2.atom_id;
235                     topset.atoms_args[atom_id].push_back(result1.link_name);
236                 }
237             } else {
238                 // Atom(...) = Link
239                 if (result2.is_link) {
240                     int atom_id = result1.atom_id;
241                     topset.atoms_args[atom_id].push_back(result2.link_name);
242                 }
243                 // Atom(...) = Atom(...)
244                 else {
245                     int atom_id1 = result1.atom_id;
246                     int atom_id2 = result2.atom_id;
247                     string orig_local_link = TopSet::get_uniq_local_link_name();
248                     topset.atoms_args[atom_id1].push_back(orig_local_link);
249                     topset.atoms_args[atom_id2].push_back(orig_local_link);
250                 }
```

```
251             }
252         }
253         // Nest
254         else {
255             if (result1.is_link) {
256                 y = tokenY, x = tokenX;
257                 syntax_error(
258                     "Top-level variable occurrence: " + result1.link_name
259                 );
260             }
261         }
262     } while (read_token(","));
263     return topset;
264 }
265
266
267 // Nest := Link | Atom | Atom '(' ')' | Atom '(' Nest {',' Nest} ')'
268 // 親子の
        Nest なら子は親へ id をかえし局所リンクを発行し、子の atom へ付け足し、親はその新規リンクを登録
269 Result_of_nest Parser::read_nest(TopSet &topset) {
270
271     // Link
272     if (is_link_initial(raw_inputs[y][x])) {
273         string cur_link = read_name();
274         read_ignore();
275         return Result_of_nest(cur_link);
276     }
277
278     // Atom | Atom '(' ')' | Atom '(' Nest {',' Nest} ')'
279     else if (is_atom_initial(raw_inputs[y][x]) || isdigit(raw_inputs[y][x])) {
280         int atom_id = topset.add_atom(read_name());
281
282         // Atom '(' ')' | Atom '(' Nest {',' Nest} ')'
283         if (read_token("(")) {
284
285             // Atom '(' ')'
286             if (read_token(")")) {
287                 return Result_of_nest(atom_id);
288             }
289
290             // Atom '(' Nest {',' Nest} ')'
291             else {
292                 do {
293                     Result_of_nest result = read_nest(topset);
294                     if (result.is_link) {
295                         topset.atoms_args[atom_id].push_back(result.link_name);
296                     } else {
297                         int child_id = result.atom_id;
298                         string orig_local_link = TopSet::get_uniq_local_link_name();
299                         topset.atoms_args[atom_id].push_back(orig_local_link);
300                         topset.atoms_args[child_id].push_back(orig_local_link);
301                     }
302                 } while (read_token(","));
303
304                 if (read_token(")")) {
305                     return Result_of_nest(atom_id);
306                 } else {
307                     syntax_error("Unexpected Token");
308                 }
309             }
310         }
311
312         // Atom
313         else {
```

```
314             return Result_of_nest(atom_id);
315         }
316
317     } else {
318         syntax_error("Unexpected Token");
319     }
320     assert(false);
321 }
322
323 // ------------------------ Guard ------------------------
324
325 // Gurad := (TypeCheck | Compare) {',' (TypeCheck | Compare)}
326 // TypeCheck := Type '(' Link ')'
327 // Type := int | unary
328 // Compare := Exp COP Exp
329 // COP := ':=' | '=\=' | '<' | '=<' | '>' | '>='
330 // AOP := '+' | '-' | '*' | '/' | 'mod'
331 // Exp := Factor { AOP Factor }
332 // Factor := Link | Num | '(' Exp ')' | ('+'|'-') Factor | 'rand' '(' Num ')'
333 // Num := [0-9]+
334
335 bool Parser::read_factor(vector<string> &tokens) {
336     if (is_link_initial(raw_inputs[y][x])) {
337         tokens.push_back(read_name());
338         return true;
339     } else if (isdigit(raw_inputs[y][x])) {
340         tokens.push_back(read_number());
341         return true;
342     } else if (read_token("(")) {
343         tokens.push_back("(");
344         if (!read_exp(tokens)) return false;
345         if (!read_token(")")) return false;
346         tokens.push_back(")");
347         return true;
348     } else if (read_token("+")) {
349         tokens.push_back("+");
350         return read_factor(tokens);
351     } else if (read_token("-")) {
352         tokens.push_back("-");
353         return read_factor(tokens);
354     } else if (read_token("rand")) {
355         tokens.push_back("rand");
356         if (!read_token("(")) return false;
357         tokens.push_back("(");
358         if (!isdigit(raw_inputs[y][x])) return false;
359         tokens.push_back(read_number());
360         if (!read_token(")")) return false;
361         tokens.push_back(")");
362         return true;
363     } else {
364         return false;
365     }
366 }
367
368 bool Parser::read_exp(vector<string> &tokens) {
369     while (true) {
370         if (!read_factor(tokens)) {
371             return false;
372         }
373         bool finish = true;
374         vector<string> arith_ops = { "+", "-", "*", "/", "mod" };
375         for (string &op : arith_ops) {
376             if (read_token(op)) {
377                 tokens.push_back(op);
378                 finish = false;
```

```
379                 break;
380             }
381         }
382         if (finish) break;
383     }
384     return true;
385 }
386
387
388 bool Parser::read_type_check(PrsGuard::TypeCheck &type_check) {
389     vector<string> types = { "int", "unary" };
390     bool ok = false;
391     for (string &type : types) {
392         if (read_token(type)) {
393             type_check.type = type;
394             ok = true;
395             break;
396         }
397     }
398     if (!ok) return false;
399     if (!read_token("(")) return false;
400     if (is_link_initial(raw_inputs[y][x])) {
401         type_check.link_name = read_name();
402     } else {
403         return false;
404     }
405     if (!read_token(")")) return false;
406     return true;
407 }
408
409
410 bool Parser::read_assign(PrsGuard::Assign &assign) {
411     if (!is_link_initial(raw_inputs[y][x])) return false;
412     assign.new_var = read_name();
413     if (!read_token("=")) return false;
414     if (!read_exp(assign.exp)) return false;
415     return true;
416 }
417
418
419 bool Parser::read_compare(PrsGuard::Compare &compare) {
420     if (!read_exp(compare.left_exp)) return false;
421     vector<string> compare_ops = { "=:=", "=\\=", "=<", "<", ">=", ">"};
422     bool ok = false;
423     for (string &op : compare_ops) {
424         if (read_token(op)) {
425             ok = true;
426             compare.op = op;
427             break;
428         }
429     }
430     if (!ok) return false;
431     if (!read_exp(compare.right_exp)) return false;
432     return true;
433 }
434
435 bool Parser::read_guard(PrsGuard &guard) {
436     do {
437         PrsGuard::TypeCheck type_check;
438         int preY = y, preX = x;
439         if (read_type_check(type_check)) {
440             guard.type_checks.push_back(type_check);
441             continue;
442         }
443         y = preY, x = preX;
```

```
444         PrsGuard::Compare compare;
445         if (read_compare(compare)) {
446             guard.compares.push_back(compare);
447             continue;
448         }
449         y = preY, x = preX;
450         PrsGuard::Assign assign;
451         if (read_assign(assign)) {
452             guard.assigns.push_back(assign);
453             continue;
454         }
455         return false;
456     } while (read_token(","));
457 
458     if (read_token("|")) {
459         return true;
460     } else {
461         return false;
462     }
463 }
464 
465 
466 // Link := [A-Z_][a-zA-Z_0-9]*
467 bool is_link_initial(char c) {
468     return isupper(c) || c == '_';
469 }
470 
471 
472 // Atom := [a-z][a-zA-Z_0-9]* | ''' .* '''
473 bool is_atom_initial(char c) {
474     return islower(c) || c == '\'';
475 }
476 
477 
478 // Atom := [a-z][a-zA-Z_0-9]* | ''' [^']+ ''' | [0-9]+
479 // Link := [A-Z_][a-zA-Z_0-9]*
480 string Parser::read_name() {
481     string name;
482     if (raw_inputs[y][x] == '\'') {
483         int tokenY = y, tokenX = x;
484         x++;
485         name += '\'';
486         while (raw_inputs[y][x] != '\'') {
487             name += raw_inputs[y][x];
488             x++;
489             if (x == (int)raw_inputs[y].size()) {
490                 y = tokenY, x = tokenX;
491                 syntax_error("Illegal Character");
492             }
493         }
494         x++;
495         name += '\'';
496     } else {
497         while (x < (int)raw_inputs[y].size()) {
498             char c = raw_inputs[y][x];
499             if (isalpha(c) || isdigit(c) || c == '_') {
500                 name += c;
501                 x++;
502             } else {
503                 break;
504             }
505         }
506     }
507     read_ignore();
508     return name;
```

```
509 }
510
511
512 // Num := [0-9]+
513 string Parser::read_number() {
514     string name;
515     while (x < (int)raw_inputs[y].size()) {
516         char c = raw_inputs[y][x];
517         if (isdigit(c)) {
518             name += c;
519             x++;
520         } else {
521             break;
522         }
523     }
524     read_ignore();
525     return name;
526 }
527
528 // -----------------------------------------------------------------------------
529
530 void TopSet::build_link_count() {
531     for (auto &args : atoms_args) {
532         for (auto &arg : args) {
533             link_count[arg]++;
534         }
535     }
536     for (auto &connect : connects) {
537         link_count[connect.first]++;
538         link_count[connect.second]++;
539     }
540 }
541
542
543 void TopSet::validate_links() {
544     for (auto &itr : link_count) {
545         const string &link_name = itr.first;
546         if (link_count[link_name] > 2) {
547             cerr << "Semantic Error: link " << link_name
548                  << " appears more than twice" << endl;
549             exit(1);
550         }
551         if (link_count[link_name] == 1) {
552             cerr << "Semantic Error: link " << link_name
553                  << " is global singleton" << endl;
554             exit(1);
555         }
556     }
557 }
558
559
560 void PrsRule::validate_links() {
561     map<string, bool> has_type;
562     for (auto &type_check : guard.type_checks) {
563         has_type[type_check.link_name] = true;
564     }
565     for (auto &assign : guard.assigns) {
566         has_type[assign.new_var] = true;
567         for (auto token : assign.exp) {
568             if (is_link_initial(token[0])) {
569                 has_type[token] = true;
570             }
571         }
572     }
573     for (auto &compare : guard.compares) {
```

```
574         for (auto token : compare.left_exp) {
575             if (is_link_initial(token[0])) {
576                 has_type[token] = true;
577             }
578         }
579         for (auto token : compare.right_exp) {
580             if (is_link_initial(token[0])) {
581                 has_type[token] = true;
582             }
583         }
584     }
585
586     for (auto &itr : head.link_count) {
587         const string &link_name = itr.first;
588         if (head.link_count[link_name] >= 3) {
589             cerr << "Semantic Error: link " << link_name
590                  << " in the head appears more than twice" << endl;
591             show();
592             exit(1);
593         }
594         if (head.link_count[link_name] == 2) {
595             if (has_type[link_name]) {
596                 cerr << "Semantic Error: link " << link_name
597                      << " in the head is local link, but it's in the guard" << endl;
598                 show();
599                 exit(1);
600             }
601         }
602         if (head.link_count[link_name] == 1){
603             if (has_type[link_name]) continue;
604             if (body.link_count[link_name] != 1) {
605                 cerr << "Semantic Error: link " << link_name
606                      << " in the head is free variable" << endl;
607                 show();
608                 exit(1);
609             }
610         }
611     }
612
613     for (auto &itr : body.link_count) {
614         const string &link_name = itr.first;
615         if (has_type[link_name]) continue;
616         if (body.link_count[link_name] >= 3) {
617             cerr << "Semantic Error: link " << link_name
618                  << " in the body appears more than twice" << endl;
619             show();
620             exit(1);
621         }
622         if (body.link_count[link_name] == 1 && head.link_count[link_name] != 1) {
623             cerr << "Semantic Error: link " << link_name
624                  << " in the body is free variable" << endl;
625             show();
626             exit(1);
627         }
628     }
629 }
630
631
632 void TopSet::build_dst_map() {
633     for (int atom_id = 0; atom_id < (int)atom_names.size(); atom_id++) {
634         int arity = atoms_args[atom_id].size();
635         // a(.., X, ..), b(.., X, ..)
636         for (int link_id = 0; link_id < arity; link_id++) {
637             string str_pair = to_string(atom_id) + ":" + to_string(link_id);
638             string &link_name = atoms_args[atom_id][link_id];
```

```
639             if (dst.count(link_name)) {
640                 dst[str_pair] = dst[link_name];
641                 dst[dst[link_name]] = str_pair;
642                 dst.erase(link_name);
643             } else {
644                 dst[str_pair] = link_name;
645                 dst[link_name] = str_pair;
646             }
647         }
648     }
649
650     // a(.., X, ..), X = Y, b(.., Y, ..)
651     for (int connect_id = 0; connect_id < (int)connects.size(); connect_id++) {
652         string &link_name1 = connects[connect_id].first;
653         string &link_name2 = connects[connect_id].second;
654         string dst_name1 = link_name1, dst_name2 = link_name2;
655         if (dst.count(link_name1)) {
656             dst_name1 = dst[link_name1];
657             dst.erase(link_name1);
658         }
659         if (dst.count(link_name2)) {
660             dst_name2 = dst[link_name2];
661             dst.erase(link_name2);
662         }
663         if (link_name1 == dst_name2 && link_name2 == dst_name1) continue;
664         dst[dst_name1] = dst_name2;
665         dst[dst_name2] = dst_name1;
666     }
667 }
668
669 pair<int, int> str_to_pair(string str) {
670     // id:pos
671     int pos = str.find(":");
672     return {stoi(str.substr(0,pos)), stoi(str.substr(pos+1))};
673 }
674
675
676 void PrsRule::build_var_id() {
677     for (auto &itr : head.link_count) {
678         const string &link_name = itr.first;
679         if (head.link_count[link_name] == 1) {
680             int id = var_id.size();
681             var_id[link_name] = id;
682         }
683     }
684     for (auto &assign : guard.assigns) {
685         int id = var_id.size();
686         var_id[assign.new_var] = id;
687     }
688 }
689
690 void set_graph(TopSet &graph) {
691
692     int atom_num = graph.atom_names.size();
693     vector<Atom*> atoms(atom_num);
694     for (int i = 0; i < atom_num; i++) {
695         Functor functor(graph.atoms_args[i].size(), graph.atom_names[i]);
696         atoms[i] = new Atom(functor);
697     }
698
699     for (int i = 0; i < atom_num; i++) {
700         Functor functor = atoms[i]->functor;
701         for (int j = 0; j < functor.arity; j++) {
702             string str_pair = to_string(i) + ":" + to_string(j);
703             pair<int,int> p = str_to_pair(graph.dst[str_pair]);
```

```
704             connect_links(atoms[i], j, atoms[p.first], p.second);
705         }
706     }
707 }
708
709
710 void set_rule(PrsRule &parser_rule) {
711
712     Rule rule;
713     int head_atom_num = parser_rule.head.atom_names.size();
714     int body_atom_num = parser_rule.body.atom_names.size();
715     vector<RuleAtom*> head_atoms(head_atom_num);
716     vector<RuleAtom*> body_atoms(body_atom_num);
717
718     for (int i = 0; i < head_atom_num; i++) {
719         Functor functor(parser_rule.head.atoms_args[i].size(), parser_rule.head.atom_names
                [i]);
720         head_atoms[i] = new RuleAtom(functor, i);
721     }
722     for (int i = 0; i < body_atom_num; i++) {
723         Functor functor(parser_rule.body.atoms_args[i].size(), parser_rule.body.atom_names
                [i]);
724         body_atoms[i] = new RuleAtom(functor, i);
725     }
726
727     for (int i = 0; i < head_atom_num; i++) {
728         for (int j = 0; j < head_atoms[i]->functor.arity; j++) {
729             string str_pair = to_string(i) + ":" + to_string(j);
730             string &dst_name = parser_rule.head.dst[str_pair];
731
732             // 自由リンク
733             if (is_link_initial(dst_name[0])) {
734                 head_atoms[i]->link[j] = RuleLink(parser_rule.var_id[dst_name]);
735             }
736             // 局所リンク
737             else {
738                 pair<int,int> p = str_to_pair(parser_rule.head.dst[str_pair]);
739                 head_atoms[i]->link[j] = RuleLink(head_atoms[p.first], p.second);
740             }
741         }
742     }
743
744     // in_guard を調べる
745     map<string, bool> in_guard;
746     for (auto &compare : parser_rule.guard.compares) {
747         for (string &token : compare.left_exp) {
748             if (parser_rule.var_id.count(token)) {
749                 in_guard[token] = true;
750             }
751         }
752         for (string &token : compare.right_exp) {
753             if (parser_rule.var_id.count(token)) {
754                 in_guard[token] = true;
755             }
756         }
757     }
758
759     for (int i = 0; i < body_atom_num; i++) {
760         Functor functor = body_atoms[i]->functor;
761         for (int j = 0; j < functor.arity; j++) {
762             string str_pair = to_string(i) + ":" + to_string(j);
763             string &dst_name = parser_rule.body.dst[str_pair];
764
765             // Guard 内のリンクも特にここでの処理はいらない？？
```

```
766             // 自由リンク
767             if (is_link_initial(dst_name[0])) {
768                 body_atoms[i]->link[j] = RuleLink(parser_rule.var_id[dst_name]);
769             }
770             // Guard にある型付きのリンク // 直したい
771             else if (in_guard[parser_rule.body.atoms_args[i][j]]) {
772                 body_atoms[i]->link[j] = RuleLink(parser_rule.var_id[parser_rule.body.
                        atoms_args[i][j]]);
773             }
774             // 局所リンク
775             else {
776                 pair<int,int> p = str_to_pair(parser_rule.body.dst[str_pair]);
777                 body_atoms[i]->link[j] = RuleLink(body_atoms[p.first], p.second);
778             }
779         }
780     }
781
782
783     // connector
784     map<string, bool> registered_connector;
785     for (auto &dst_itr : parser_rule.body.dst) {
786         const string &link_name1 = dst_itr.first;
787         const string &link_name2 = dst_itr.second;
788         if (isdigit(link_name1[0])) continue;
789         if (isdigit(link_name2[0])) continue;
790         if (registered_connector[link_name1]) continue;
791         if (registered_connector[link_name2]) continue;
792         rule.connectors.push_back({parser_rule.var_id[link_name1], parser_rule.var_id[
                link_name2]});
793         registered_connector[link_name1] = true;
794         registered_connector[link_name2] = true;
795     }
796
797     // 以下、いずれ修正
798
799     for (auto &type_check : parser_rule.guard.type_checks) {
800         type_check.link_name = "#" + to_string(parser_rule.var_id[type_check.link_name]);
801     }
802     for (auto &compare : parser_rule.guard.compares) {
803         for (string &token : compare.left_exp) {
804             if (parser_rule.var_id.count(token)) {
805                 token = "#" + to_string(parser_rule.var_id[token]);
806             }
807         }
808         for (string &token : compare.right_exp) {
809             if (parser_rule.var_id.count(token)) {
810                 token = "#" + to_string(parser_rule.var_id[token]);
811             }
812         }
813     }
814
815     // rule.guard = parser_rule.guard;
816     for (auto &type_check : parser_rule.guard.type_checks) {
817         rule.guard.type_checks.emplace_back(type_check.link_name, type_check.type);
818     }
819     for (auto &compare : parser_rule.guard.compares) {
820         rule.guard.compares.emplace_back(compare.left_exp, compare.op, compare.right_exp);
821     }
822     rule.head_atoms = head_atoms;
823     rule.body_atoms = body_atoms;
824     rule.freelink_num = parser_rule.var_id.size();
825     rulelist.push_back(rule);
826 }
827
```

```
828
829 void Parser::parse() {
830     read_istream();
831     read_sentences();
832     // show();
833
834     graph.build_dst_map();
835     graph.build_link_count();
836     graph.validate_links();
837
838     for (auto &prsrule : rules) {
839         prsrule.head.build_dst_map();
840         prsrule.body.build_dst_map();
841
842         prsrule.head.build_link_count();
843         prsrule.body.build_link_count();
844
845         prsrule.build_var_id();
846
847         prsrule.validate_links();
848     }
849 }
850
851
852 void load(Parser &parser) {
853     set_graph(parser.graph);
854     for (auto &prsrule : parser.rules) {
855         set_rule(prsrule);
856     }
857 }
```

ソースコード A.5　rule.hpp

```
1  #ifndef RULE_HPP
2  #define RULE_HPP
3
4  #include <iostream>
5  using std::cin;
6  using std::cout;
7  using std::cerr;
8  using std::endl;
9  using std::ostream;
10 using std::getline;
11 #include <vector>
12 using std::vector;
13 #include <string>
14 using std::string;
15 #include <map>
16 using std::map;
17 using std::pair;
18 #include <unordered_set>
19 using std::unordered_set;
20 #include <unordered_map>
21 using std::unordered_map;
22 #include <list>
23 using std::list;
24 #include <algorithm>
25 using std::max;
26 using std::min;
27 #include <random>
28 #include <cassert>
29
30 #define debug(x) cerr << (#x) << ": " << (x) << endl;
```

```
31
32 class Functor;
33 class Atom;
34 class Link;
35 class RuleAtom;
36 class RuleLink;
37 class Guard;
38 class Register;
39 class Rule;
40
41 extern map<Functor,list<Atom*>> atomlist;
42 extern vector<Rule> rulelist;
43
44 class Functor {
45   public:
46     int arity;
47     string name;
48
49     Functor();
50     ~Functor();
51     Functor(int arity_, string name_);
52
53     bool operator==(const Functor &rhs) const;
54     bool operator!=(const Functor &rhs) const;
55     bool is_int() const;
56     bool operator<(const Functor &rhs) const;
57     friend ostream& operator<<(ostream& ost, const Functor &rhs);
58
59 };
60
61
62 class Link {
63   public:
64     Atom *atom;
65     int pos;
66
67     Link();
68     ~Link();
69     Link(Atom *atom_, int pos_);
70
71     bool operator<(const Link &rhs) const;
72 };
73
74 class Atom {
75   public:
76     Functor functor;
77     vector<Link> link;
78     list<Atom*>::iterator itr;
79
80     Atom();
81     ~Atom();
82     Atom(Functor functor_);
83
84     bool is_int();
85 };
86
87 class RuleLink {
88   public:
89     RuleAtom *atom;
90     int pos;
91
92     RuleLink();
93     ~RuleLink();
94     RuleLink(int pos_);
95     RuleLink(RuleAtom *atom_, int pos_);
```

```
 96
 97     bool is_freelink();
 98     int freelinkID();
 99 };
100
101
102 class RuleAtom {
103   public:
104     Functor functor;
105     int id;
106     vector<RuleLink> link;
107
108     RuleAtom();
109     ~RuleAtom();
110     RuleAtom(Functor functor_, int id_);
111     RuleAtom(Functor functor_, int id_, vector<RuleLink> link_);
112 };
113
114
115 class Guard {
116   public:
117
118     class Compare {
119       public:
120
121         vector<string> left_exp;
122         string op;
123         vector<string> right_exp;
124
125         Compare();
126         ~Compare();
127         Compare(vector<string> &left_exp_, string &op_, vector<string> &right_exp_);
128
129         bool is_null();
130     };
131
132
133     class TypeCheck {
134       public:
135         string link;
136         string type;
137
138         TypeCheck();
139         ~TypeCheck();
140         TypeCheck(string link_, string type_);
141
142         bool is_null();
143     };
144
145     class Assign {
146       public:
147         string new_var;
148         vector<string> exp;
149
150         Assign();
151         ~Assign();
152
153         bool is_null();
154     };
155
156     vector<TypeCheck> type_checks;
157     vector<Compare> compares;
158     vector<Assign> assigns;
159
160     Guard();
```

```
161     ~Guard();
162
163     bool is_null();
164 };
165
166
167 class Rule {
168   public:
169     int freelink_num;
170     vector<RuleAtom*> head_atoms;
171     Guard guard;
172     vector<RuleAtom*> body_atoms;
173     vector<pair<int,int>> connectors;
174
175     Rule();
176     ~Rule();
177
178     void show();
179 };
180
181
182 class Register {
183   public:
184     vector<Atom*> head_atoms;
185     vector<Atom*> body_atoms;
186     vector<Link> freelinks;
187     map<int,string> expected_unary;
188
189     Register();
190     ~Register();
191     Register(Rule &rule);
192 };
193
194 void connect_links(Atom *atom1, int pos1, Atom *atom2, int pos2);
195 bool try_rule(Rule &rule);
196 bool find_atom(Rule &rule, Register &reg);
197 bool set_atom_to_reg(Rule &rule, Register &reg, Atom* atom, int head_id);
198 void remove_atom_from_reg(Rule &rule, Register &reg, Atom* atom, int head_id);
199 int eval_exp(Rule &rule, Register &reg, vector<string> &tokens, int &i);
200 int eval_factor(Rule &rule, Register &reg, vector<string> &tokens, int &i);
201 int eval_term(Rule &rule, Register &reg, vector<string> &tokens, int &i);
202 int eval_exp(Rule &rule, Register &reg, vector<string> &tokens, int &i);
203 bool eval_compare(int left_exp, string &op, int right_exp);
204 bool guard_check(Rule &rule, Register &reg);
205 void rewrite(Rule &rule, Register &reg);
206
207 void show_graph();
208 void nest_dump(Atom* atom, int depth, unordered_set<Atom*> &dumped_atoms, map<Link, int>
        &locallink_id);
209 void dump();
210 void show_atomlist_size();
211 void show_rules();
212
213 #endif
```

ソースコード A.6　rule.cpp

```
1 #include "rule.hpp"
2 #include "parser.hpp"
3
4 map<Functor,list<Atom*>> atomlist;
5 vector<Rule> rulelist;
6 map<Functor, map<int, unordered_map<string, list<Atom*>>>> unary_indexed_atomlist;
```

```
 7 // unary_indexed_atomlist[dst_functor][dst_pos][unary_name] = {*dst_atom, ...}
 8
 9 unordered_map<Atom*, list<Atom*>::iterator> unary_indexed_atom_itr;
10 // unary_indexed_atom_itr[*unary_atom] = unary_indexed_atomlist::iterator<*dst_atom>
11
12 Functor::Functor() {}
13 Functor::~Functor() {}
14
15 Functor::Functor(int arity_, string name_): arity(arity_), name(name_) {}
16
17 bool Functor::operator==(const Functor &rhs) const {
18     return arity == rhs.arity && name == rhs.name;
19 }
20
21 bool Functor::operator!=(const Functor &rhs) const {
22     return arity != rhs.arity || name != rhs.name;
23 }
24
25 bool Functor::is_int() const {
26     return isdigit(name[0]);
27 }
28
29 bool Functor::operator<(const Functor &rhs) const {
30     return name != rhs.name ? name < rhs.name : arity < rhs.arity;
31 }
32
33 ostream& operator<<(ostream& ost, const Functor &rhs) {
34     ost << "<" << rhs.name << "," << rhs.arity << ">";
35     return ost;
36 }
37
38
39 Link::Link(): atom(NULL), pos(0) {}
40 Link::~Link() {}
41
42 Link::Link(Atom *atom_, int pos_): atom(atom_), pos(pos_) {}
43
44 bool Link::operator<(const Link &rhs) const {
45     return atom != rhs.atom ? atom < rhs.atom : pos < rhs.pos;
46 }
47
48
49 Atom::Atom() {}
50
51 Atom::~Atom() {
52     assert(*itr == this);
53     atomlist[functor].erase(itr);
54 }
55
56 Atom::Atom(Functor functor_): functor(functor_) {
57     link.resize(functor.arity);
58     atomlist[functor].push_front(this);
59     itr = atomlist[functor].begin();
60 }
61
62 bool Atom::is_int() {
63     return functor.arity == 1 && isdigit(functor.name[0]);
64 }
65
66 // atom1 の pos1 に atom2 の pos2 をつなぐ
67 void connect_links(Atom *atom1, int pos1, Atom *atom2, int pos2) {
68     // atom1 の pos1 にあった Link が unary
69     Atom *dst_atom = atom1->link[pos1].atom;
70     if (dst_atom != NULL && dst_atom->functor.arity == 1) {
```

```
71          unary_indexed_atomlist[atom1->functor][pos1][dst_atom->functor.name].erase(
72              unary_indexed_atom_itr[dst_atom]
73          );
74      }
75
76      // atom1 の pos1 に atom2 の pos2 をつなぐ
77      atom1->link[pos1].atom = atom2;
78      atom1->link[pos1].pos = pos2;
79
80      // atom2 が unary
81      if (atom2->functor.arity == 1) {
82          unary_indexed_atomlist[atom1->functor][pos1][atom2->functor.name].emplace_front(
                atom1);
83          unary_indexed_atom_itr[atom2]
84              = unary_indexed_atomlist[atom1->functor][pos1][atom2->functor.name].begin();
85      }
86  }
87
88
89
90
91  RuleLink::RuleLink() {}
92  RuleLink::~RuleLink() {}
93
94  RuleLink::RuleLink(int pos_): atom(NULL), pos(pos_) {}
95
96  RuleLink::RuleLink(RuleAtom *atom_, int pos_): atom(atom_), pos(pos_) {}
97
98  bool RuleLink::is_freelink() {
99      return atom == NULL;
100 }
101
102 int RuleLink::freelinkID() {
103     assert(is_freelink());
104     return pos;
105 }
106
107
108 RuleAtom::RuleAtom() {}
109 RuleAtom::~RuleAtom() {}
110
111 RuleAtom::RuleAtom(Functor functor_, int id_): functor(functor_), id(id_) {
112     link.resize(functor.arity);
113 }
114
115 RuleAtom::RuleAtom(Functor functor_, int id_, vector<RuleLink> link_):
116     functor(functor_), id(id_), link(link_) {}
117
118
119 Guard::Compare::Compare() {}
120 Guard::Compare::~Compare() {}
121
122 Guard::Compare::Compare(vector<string> &left_exp_, string &op_, vector<string> &
        right_exp_):
123     left_exp(left_exp_), op(op_), right_exp(right_exp_) {}
124
125 bool Guard::Compare::is_null() {
126     return op == "";
127 }
128
129
130 Guard::TypeCheck::TypeCheck() {}
131 Guard::TypeCheck::~TypeCheck() {}
132
```

```
133 Guard::TypeCheck::TypeCheck(string link_, string type_): link(link_), type(type_) {}
134
135 bool Guard::TypeCheck::is_null() {
136     return type == "";
137 }
138
139
140 Guard::Assign::Assign() {}
141 Guard::Assign::~Assign() {}
142
143
144 Guard::Guard() {}
145 Guard::~Guard() {}
146
147 bool Guard::is_null() {
148     return type_checks.empty() && compares.empty();
149 }
150
151
152 Rule::Rule(): freelink_num(0) {}
153 Rule::~Rule() {}
154
155 void Rule::show() {
156     cout << "----------- head -----------" << endl;
157     for (int i = 0; i < (int)head_atoms.size(); i++) {
158         cout << head_atoms[i]->functor << " [" << head_atoms[i] << "] (";
159         int arity = (int)head_atoms[i]->functor.arity;
160         for (int j = 0; j < arity; j++) {
161             cout << ((head_atoms[i]->link[j].is_freelink()) ? "FL:" : "LL:")
162                  << head_atoms[i]->link[j].atom << "(" << head_atoms[i]->link[j].pos << ")"
163                  << (j + 1 == arity ? "" : ", ");
164         }
165         cout << ")" << endl;
166     }
167     cout << "----------- body -----------" << endl;
168     for (int i = 0; i < (int)body_atoms.size(); i++) {
169         cout << body_atoms[i]->functor << " [" << body_atoms[i] << "] (";
170         int arity = (int)body_atoms[i]->functor.arity;
171         for (int j = 0; j < arity; j++) {
172             cout << ((body_atoms[i]->link[j].is_freelink()) ? "FL:" : "LL:")
173                  << body_atoms[i]->link[j].atom << "(" << body_atoms[i]->link[j].pos << ")"
174                  << (j + 1 == arity ? "" : ", ");
175         }
176         cout << ")" << endl;
177     }
178     for (auto &p : connectors) {
179         cout << p.first << " = " << p.second << endl;
180     }
181 }
182
183
184 Register::Register() {}
185 Register::~Register() {}
186
187 Register::Register(Rule &rule) {
188     head_atoms.resize(rule.head_atoms.size());
189     body_atoms.resize(rule.body_atoms.size());
190     freelinks.resize(rule.freelink_num);
191 }
192
193
194 long long num_rules_success = 0;
195 bool try_rule(Rule &rule) {
196     Register reg(rule);
197     if (find_atom(rule,reg)) {
```

```
198         rewrite(rule,reg);
199         num_rules_success++;
200         return true;
201     } else {
202         return false;
203     }
204 }
205
206 /*
207     ルール内の未決定のアトムを 1つ選び、
208     それにマッチするアトムをグラフの中から 1つ選びレジスタに格納する
209 */
210
211 list<Atom*>* get_unary_indexed_atomlist(
212     Functor &dst_functor, int dst_pos, string &unary_name)
213 {
214     if (unary_indexed_atomlist[dst_functor][dst_pos].count(unary_name) == 0) {
215         return NULL;
216     }
217     return &unary_indexed_atomlist[dst_functor][dst_pos][unary_name];
218 }
219
220 // [a_0, ... , a_k-1, a_k (itr), ... , a_n-1]
221 // ->[a_k (itr), ... , a_n-1, a_0, ... , a_k-1]
222 void splice(list<Atom*> &a, list<Atom*>::iterator &itr) {
223     a.splice(a.end(), a, a.begin(), itr);
224 }
225
226 long long back_track = 0;
227
228 bool find_atom(Rule &rule, Register &reg) {
229     // 未決定アトムの中で最小のアトムリストのものを選ぶ
230     int head_id = -1;
231     list<Atom*> *start_point_list = NULL;
232     if (!guard_check(rule, reg)) return false;
233
234     for (int i = 0; i < (int)reg.head_atoms.size(); i++) {
235         if (reg.head_atoms[i] != NULL) continue;
236
237         list<Atom*> *atomlist_i = &atomlist[rule.head_atoms[i]->functor];
238         if (start_point_list == NULL ||
239             start_point_list->size() > atomlist_i->size())
240         {
241             start_point_list = atomlist_i;
242             head_id = i;
243         }
244
245         int arity = rule.head_atoms[i]->functor.arity;
246         for (int j = 0; j < arity; j++) {
247             if (!rule.head_atoms[i]->link[j].is_freelink()) continue;
248             int fid = rule.head_atoms[i]->link[j].freelinkID();
249             if (reg.expected_unary.count(fid)) {
250                 atomlist_i = get_unary_indexed_atomlist(rule.head_atoms[i]->functor, j, reg
                        .expected_unary[fid]);
251                 if (atomlist_i == NULL) {
252                     return false;
253                 }
254                 if (start_point_list->size() > atomlist_i->size()) {
255                     start_point_list = atomlist_i;
256                     head_id = i;
257                 }
258             }
259         }
260     }
```

```
261
262
263     if (start_point_list == NULL) {
264         return true;
265     } else {
266         for (auto itr = start_point_list->begin(); itr != start_point_list->end(); ++itr)
                {
267             Atom* atom = *itr;
268             bool ok = set_atom_to_reg(rule, reg, atom, head_id);
269             if (ok) {
270                 if (find_atom(rule, reg)) {
271                     splice(*start_point_list, itr);
272                     return true;
273                 } else {
274                     remove_atom_from_reg(rule, reg, atom, head_id);
275                 }
276             }
277             back_track++;
278         }
279         return false;
280     }
281 }
282
283
284 /*
285     reg.head_atoms[head_id] に atom をいれて問題がないか検査
286     問題がなければ reg にいれて true を返す
287     問題があれば reg から外して false を返す
288     再帰的に連結グラフのアトムを検査
289 */
290
291 bool set_atom_to_reg(Rule &rule, Register &reg, Atom* atom, int head_id) {
292
293     // 他でreg 登録済みのアトム
294     for (Atom *registered_atom : reg.head_atoms) {
295         if (atom == registered_atom) return false;
296     }
297
298     reg.head_atoms[head_id] = atom;
299     RuleAtom *rule_atom = rule.head_atoms[head_id];
300     /* リンクが合っているか */
301     int arity = atom->functor.arity;
302     for (int i = 0; i < arity; i++) {
303         // 自由リンク
304         if (rule_atom->link[i].is_freelink()) {
305             continue;
306         }
307         Atom* dst_atom = atom->link[i].atom;
308
309         // リンク先のファンクタが違う
310         if (dst_atom->functor != rule_atom->link[i].atom->functor) {
311             reg.head_atoms[head_id] = NULL;
312             return false;
313         }
314         // 接続場所が合っているか
315         int dst_pos = rule_atom->link[i].pos;
316         if (dst_atom->link[dst_pos].atom != atom ||
317             dst_atom->link[dst_pos].pos != i) {
318             reg.head_atoms[head_id] = NULL;
319             return false;
320         }
321     }
322     /* リンクをレジスタに登録 */
323     for (int i = 0; i < arity; i++) {
```

```
324         // 自由リンク
325         if (rule_atom->link[i].is_freelink()) {
326             reg.freelinks[rule_atom->link[i].freelinkID()] = atom->link[i];
327             continue;
328         }
329 
330         Atom* dst_atom = atom->link[i].atom;
331         int dst_id = rule_atom->link[i].atom->id; // ルールでは来るはずのid
332         // 自由リンクでない かつ レジスタ [dst_id]は未登録
333         if (reg.head_atoms[dst_id] == NULL) {
334             bool ok = set_atom_to_reg(rule, reg, dst_atom, dst_id);
335             if (!ok) {
336                 reg.head_atoms[head_id] = NULL;
337                 return false;
338             }
339         }
340         // 自由リンクでない かつ レジスタ [dst_id]は登録済み
341         else if (reg.head_atoms[dst_id] != NULL) {
342             if (reg.head_atoms[dst_id]->link[rule_atom->link[i].pos].atom != atom) {
343                 reg.head_atoms[head_id] = NULL;
344                 return false;
345             }
346         }
347     }
348 
349     return true;
350 }
351 
352 void remove_atom_from_reg(Rule &rule, Register &reg, Atom* atom, int head_id) {
353     Functor functor = atom->functor;
354     reg.head_atoms[head_id] = NULL;
355 
356     int arity = functor.arity;
357     for (int i = 0; i < arity; i++) {
358 
359         // 自由リンク
360         if (rule.head_atoms[head_id]->link[i].is_freelink()) {
361             int fi = rule.head_atoms[head_id]->link[i].freelinkID();
362             reg.freelinks[fi] = Link(NULL, 0);
363             continue;
364         }
365 
366         Atom* dst_atom = atom->link[i].atom;
367         int dst_id = rule.head_atoms[head_id]->link[i].atom->id;
368         // 自由リンクでない かつ レジスタ登録済み
369         if (reg.head_atoms[dst_id] != NULL) {
370             remove_atom_from_reg(rule, reg, dst_atom, dst_id);
371         }
372     }
373 }
374 
375 
376 /*
377     TOP := '+' | '-'
378     FOP := '*' | '/' | 'mod'
379     Exp := Term { TOP Term }
380     Term := Factor { FOP Factor }
381     Factor := Link | Num | '(' Exp ')' | TOP Factor
382     Num := [0-9]+
383 */
384 
385 int eval_factor(Rule &rule, Register &reg, vector<string> &tokens, int &i) {
386     if (tokens[i][0] == '#') {
387         int freelink_id = std::stoi(tokens[i].substr(1));
```

```
388         i++;
389         return std::stoi(reg.freelinks[freelink_id].atom->functor.name);
390     }
391 
392     if (isdigit(tokens[i][0])) {
393         int result = std::stoi(tokens[i]);
394         i++;
395         return result;
396     }
397 
398     if (tokens[i] == "(") {
399         i++;
400         int exp = eval_exp(rule, reg, tokens, i);
401         assert(tokens[i] == ")");
402         i++;
403         return exp;
404     }
405 
406     if (tokens[i] == "+") {
407         i++;
408         return eval_factor(rule, reg, tokens, i);
409     }
410 
411     if (tokens[i] == "-") {
412         i++;
413         return - eval_factor(rule, reg, tokens, i);
414     }
415 
416     if (tokens[i] == "rand") {
417         i++;
418         assert(tokens[i] == "(");
419         i++;
420         int rand_max = std::stoi(tokens[i]);
421         assert(tokens[i] == ")");
422         i++;
423 
424         static std::random_device seed_gen;
425         static std::mt19937 mt(seed_gen());
426         std::uniform_int_distribution<> rand_int(0, rand_max-1);
427         return rand_int(mt);
428     }
429 
430     assert(false);
431 }
432 
433 int eval_term(Rule &rule, Register &reg, vector<string> &tokens, int &i) {
434     int result = eval_factor(rule, reg, tokens, i);
435     while (true) {
436         if ((int)tokens.size() == i) break;
437         if (tokens[i] != "*" && tokens[i] != "/") break;
438         string op = tokens[i];
439         i++;
440         int term = eval_factor(rule, reg, tokens, i);
441         if (op == "*") {
442             result *= term;
443         } else if (op == "/") {
444             result /= term;
445         } else {
446             assert(false);
447         }
448     }
449     return result;
450 }
451 
452 int eval_exp(Rule &rule, Register &reg, vector<string> &tokens, int &i) {
```

```
453     int result = eval_term(rule, reg, tokens, i);
454     while (true) {
455         if ((int)tokens.size() == i) break;
456         if (tokens[i] != "+" && tokens[i] != "-") break;
457         string &op = tokens[i];
458         i++;
459         int term = eval_term(rule, reg, tokens, i);
460         if (op == "+") {
461             result += term;
462         } else if (op == "-") {
463             result -= term;
464         } else {
465             assert(false);
466         }
467     }
468     return result;
469 }
470 
471 bool eval_compare(int left_exp, string &op, int right_exp) {
472     if (op == "=:=") return left_exp == right_exp;
473     if (op == "=\\=") return left_exp != right_exp;
474     if (op == "=<") return left_exp <= right_exp;
475     if (op == "<") return left_exp < right_exp;
476     if (op == ">=") return left_exp >= right_exp;
477     if (op == ">") return left_exp > right_exp;
478     assert(false);
479 }
480 
481 bool vars_in_exp_are_completed(Register &reg, vector<string> &tokens) {
482     for (string &token : tokens) {
483         if (token[0] == '#') {
484             int freelink_id = std::stoi(token.substr(1));
485             if (reg.freelinks[freelink_id].atom == NULL) {
486                 return false;
487             }
488         }
489     }
490     return true;
491 }
492 
493 
494 
495 // そこまでで評価ができそうなGuard文の検査
496 // expected_unary のチェックも行う
497 bool guard_check(Rule &rule, Register &reg) {
498 
499     for (auto &compare : rule.guard.compares) {
500         // TODO: リンクがすべてint であることを確認
501 
502         bool left_exp_is_completed = vars_in_exp_are_completed(reg, compare.left_exp);
503         bool right_exp_is_completed = vars_in_exp_are_completed(reg, compare.right_exp);
504         if (!left_exp_is_completed && !right_exp_is_completed) {
505             continue;
506         }
507         int i, left_exp, right_exp;
508         if (left_exp_is_completed) {
509             i = 0;
510             left_exp = eval_exp(rule, reg, compare.left_exp, i);
511         }
512         if (right_exp_is_completed) {
513             i = 0;
514             right_exp = eval_exp(rule, reg, compare.right_exp, i);
515         }
516         if (left_exp_is_completed && !right_exp_is_completed && compare.op == "=:=" &&
```

```
517             compare.right_exp.size() == 1 && compare.right_exp[0][0] == '#')
518         {
519             int freelink_id = std::stoi(compare.right_exp[0].substr(1));
520             reg.expected_unary[freelink_id] = to_string(left_exp);
521         }
522         if (!left_exp_is_completed && right_exp_is_completed && compare.op == ":=" &&
523             compare.left_exp.size() == 1 && compare.left_exp[0][0] == '#')
524         {
525             int freelink_id = std::stoi(compare.left_exp[0].substr(1));
526             reg.expected_unary[freelink_id] = to_string(right_exp);
527         }
528         if (left_exp_is_completed && right_exp_is_completed &&
529             !eval_compare(left_exp, compare.op, right_exp))
530         {
531             return false;
532         }
533     }
534     return true;
535 }
536
537
538 void rewrite(Rule &rule, Register &reg) {
539     for (Atom* &atom : reg.head_atoms) {
540         assert(atom != NULL);
541     }
542 /*
543     freelink が head_atoms を指しているとき、
544     そのfreelink を新しいほうにつなぎ変えてからもとの head_atoms を消す必要がある
545 */
546
547     vector<bool> in_guard(rule.freelink_num);
548     for (auto &compare : rule.guard.compares) {
549         for (string &token : compare.left_exp) {
550             if (token[0] == '#') {
551                 int freelink_id = std::stoi(token.substr(1));
552                 in_guard[freelink_id] = true;
553             }
554         }
555         for (string &token : compare.right_exp) {
556             if (token[0] == '#') {
557                 int freelink_id = std::stoi(token.substr(1));
558                 in_guard[freelink_id] = true;
559             }
560         }
561     }
562     map<pair<int,int>, Atom*> typed_atoms; // <body_atoms_id, freelink_id>
563
564
565     const int rule_body_size = rule.body_atoms.size();
566
567     // 1. 新しいアトムを生成する
568     for (int i = 0; i < rule_body_size; i++) {
569         reg.body_atoms[i] = new Atom(rule.body_atoms[i]->functor);
570     }
571
572     // 2. 自由リンクの先からこちらを向いてるものを新しいアトムに指す
573     for (int i = 0; i < rule_body_size; i++) {
574         const int arity = rule.body_atoms[i]->functor.arity;
575         for (int j = 0; j < arity; j++) {
576             // body の i 番目のアトムの j 番目のリンクについて
577
578             if (rule.body_atoms[i]->link[j].is_freelink()) {
579                 int fi = rule.body_atoms[i]->link[j].freelinkID();
580                 if (in_guard[fi]) {
```

```
581                 typed_atoms[{i, fi}] = new Atom(reg.freelinks[fi].atom->functor);
582                 connect_links(typed_atoms[{i, fi}], 0, reg.body_atoms[i], j);
583             } else {
584                 // int dst_pos = reg.freelinks[fi].pos;
585                 // reg.freelinks[fi].atom->link[dst_pos].atom = reg.body_atoms[i];
586                 // reg.freelinks[fi].atom->link[dst_pos].pos = j;
587                 connect_links(reg.freelinks[fi].atom, reg.freelinks[fi].pos, reg.
                        body_atoms[i], j);
588             }
589         }
590     }
591 }
592
593 // 3. bodyatom から自由リンクを指す & 局所リンクを繋ぐ
594 for (int i = 0; i < rule_body_size; i++) {
595     const int arity = rule.body_atoms[i]->functor.arity;
596     for (int j = 0; j < arity; j++) {
597         // body の i 番目のアトムの j 番目のリンクについて
598         if (rule.body_atoms[i]->link[j].is_freelink()) {
599             int fi = rule.body_atoms[i]->link[j].freelinkID();
600             if (in_guard[fi]) {
601                 connect_links(reg.body_atoms[i], j, typed_atoms[{i, fi}], 0);
602             } else {
603                 connect_links(reg.body_atoms[i], j, reg.freelinks[fi].atom, reg.
                        freelinks[fi].pos);
604             }
605             // reg.body_atoms[i]->link[j] = reg.freelinks[fi];
606         } else {
607             int li = rule.body_atoms[i]->link[j].atom->id;
608             // reg.body_atoms[i]->link[j].atom = reg.body_atoms[li];
609             // reg.body_atoms[i]->link[j].pos = rule.body_atoms[i]->link[j].pos;
610             connect_links(reg.body_atoms[i], j, reg.body_atoms[li], rule.body_atoms[i
                    ]->link[j].pos);
611         }
612     }
613 }
614
615 // 4. 自由リンク同士の接続
616 for (pair<int,int> &p : rule.connectors) {
617     int u = p.first;
618     int v = p.second;
619     // reg.freelinks[u].atom->link[reg.freelinks[u].pos] = reg.freelinks[v];
620     connect_links(reg.freelinks[u].atom, reg.freelinks[u].pos, reg.freelinks[v].atom,
            reg.freelinks[v].pos);
621     // reg.freelinks[v].atom->link[reg.freelinks[v].pos] = reg.freelinks[u];
622     connect_links(reg.freelinks[v].atom, reg.freelinks[v].pos, reg.freelinks[u].atom,
            reg.freelinks[u].pos);
623 }
624
625 for (int i = 0; i < rule.freelink_num; i++) {
626     if (in_guard[i]) {
627         Atom *dst_unary_atom = reg.freelinks[i].atom;
628         Atom *h_atom = dst_unary_atom->link[0].atom;
629         unary_indexed_atomlist[h_atom->functor][dst_unary_atom->link[0].pos][
                dst_unary_atom->functor.name].erase(
630             unary_indexed_atom_itr[dst_unary_atom]
631         );
632         delete dst_unary_atom;
633     }
634 }
635
636 // 5. headatom を消す
637 const int rule_head_size = rule.head_atoms.size();
638 for (int i = 0; i < rule_head_size; i++) {
```

```
639         if (reg.head_atoms[i]->functor.arity == 1 && !rule.head_atoms[i]->link[0].
                is_freelink()) {
640             int dst_id = rule.head_atoms[i]->link[0].atom->id;
641             int dst_pos = rule.head_atoms[i]->link[0].pos;
642             unary_indexed_atomlist[rule.head_atoms[dst_id]->functor][dst_pos][rule.
                    head_atoms[i]->functor.name].erase(
643                 unary_indexed_atom_itr[reg.head_atoms[i]]
644             );
645         }
646         delete reg.head_atoms[i];
647     }
648 }
649
650 /* ---------------------------- dump ---------------------------- */
651
652 void show_graph() {
653     map<pair<Atom*, int>, string> mp;
654     for (auto al : atomlist) {
655         Functor functor = al.first;
656         auto &list = al.second;
657         cout << functor << " size = " << list.size() << endl;
658     }
659 }
660
661 void nest_dump(Atom* atom, int depth, unordered_set<Atom*> &dumped_atoms, map<Link, int>
        &locallink_id) {
662
663     dumped_atoms.insert(atom);
664     Functor &functor = atom->functor;
665     cout << functor.name;
666     int args = functor.arity + (depth > 0 ? -1 : 0);
667     if (args == 0) return;
668     cout << "(";
669     for (int i = 0; i < args; i++) {
670         if (locallink_id.find(atom->link[i]) != locallink_id.end()) {
671             cout << "L" << locallink_id[atom->link[i]];
672         } else {
673             Atom *dst_atom = atom->link[i].atom;
674             int dst_pos = atom->link[i].pos;
675             if (dst_atom->functor.arity == dst_pos + 1 &&
676                 depth < 10000 &&
677                 dumped_atoms.find(dst_atom) == dumped_atoms.end())
678             {
679                 nest_dump(dst_atom, depth+1, dumped_atoms, locallink_id);
680             } else {
681                 int id = locallink_id.size();
682                 locallink_id[Link(atom, i)] = id;
683                 cout << "L" << id;
684             }
685         }
686         if (i < args-1) cout << ",";
687     }
688     cout << ")";
689 }
690
691
692 void dump() {
693     vector<pair<int,Functor>> size_functor_pairs;
694     for (auto &itr : atomlist) {
695         size_functor_pairs.push_back({itr.second.size(), itr.first});
696     }
697     sort(size_functor_pairs.begin(), size_functor_pairs.end());
698     unordered_set<Atom*> dumped_atoms;
699     map<Link, int> locallink_id;
700     for (pair<int,Functor> &p : size_functor_pairs) {
```

```
701         Functor &functor = p.second;
702         if (functor.is_int()) continue;
703         for (Atom* atom : atomlist[functor]) {
704             if (dumped_atoms.find(atom) == dumped_atoms.end()) {
705                 nest_dump(atom, 0, dumped_atoms, locallink_id);
706                 cout << ". ";
707             }
708         }
709     }
710     cout << endl;
711 }
712 
713 void show_atomlist_size() {
714     cout << "------------ atom list size ------------" << endl;
715     for (auto &al : atomlist) {
716         cout << al.first << " size = " << al.second.size() << endl;
717     }
718 }
719 
720 void show_rules() {
721     int R = rulelist.size();
722     for (int i = 0; i < R; i++) {
723         printf("------------ Rule %d ------------\n", i);
724         rulelist[i].show();
725     }
726 }
727 
728 int main(void) {
729     Parser parser;
730     parser.parse();
731     load(parser); // vm.load(parser);
732     // show_rules();
733 
734     while (true) {
735         bool success = false;
736         for (Rule &rule : rulelist) {
737             if (try_rule(rule)) {
738                 success = true;
739                 break;
740             }
741         }
742         if (!success) break;
743     }
744     // debug(num_rules_success);
745     dump();
746     debug(back_track);
747     return 0;
748 }
```